```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity buton_paraziti is
   port(
      a : in STD_LOGIC;
      led : out STD_LOGIC;
      clk: in STD_LOGIC
      );
end buton_paraziti;

architecture buton_paraziti of buton_paraziti is
   signal tasta: STD_LOGIC:='1';

begin

   process (clk)
      variable Qint: STD_LOGIC_VECTOR (19 downto 0):=(others =>'0');
      variable val_veche: STD_LOGIC:='0';
   begin
      if (CLK'event and CLK='1') then
         if (a xor val_veche)='1' then
            Qint:=(others=>'0');
            val_veche:=a;
         else
            Qint:=Qint+'1';
            if ((Qint="00111111111111111111") and ((val_veche xor a)='0')) THEN
               tasta <= val_veche;
            end if;
         end if;
      end if;
   end process;

   process (clk)
      variable temp: std_logic:='0';
      variable tasta_veche: std_logic:='1';
   begin
      if (CLK'event and CLK='1') then
         if ( tasta='0' and tasta_veche='1') then
            temp:=not temp;
         end if;
         tasta_veche :=tasta;
      end if;
         led <=temp;
   end process;

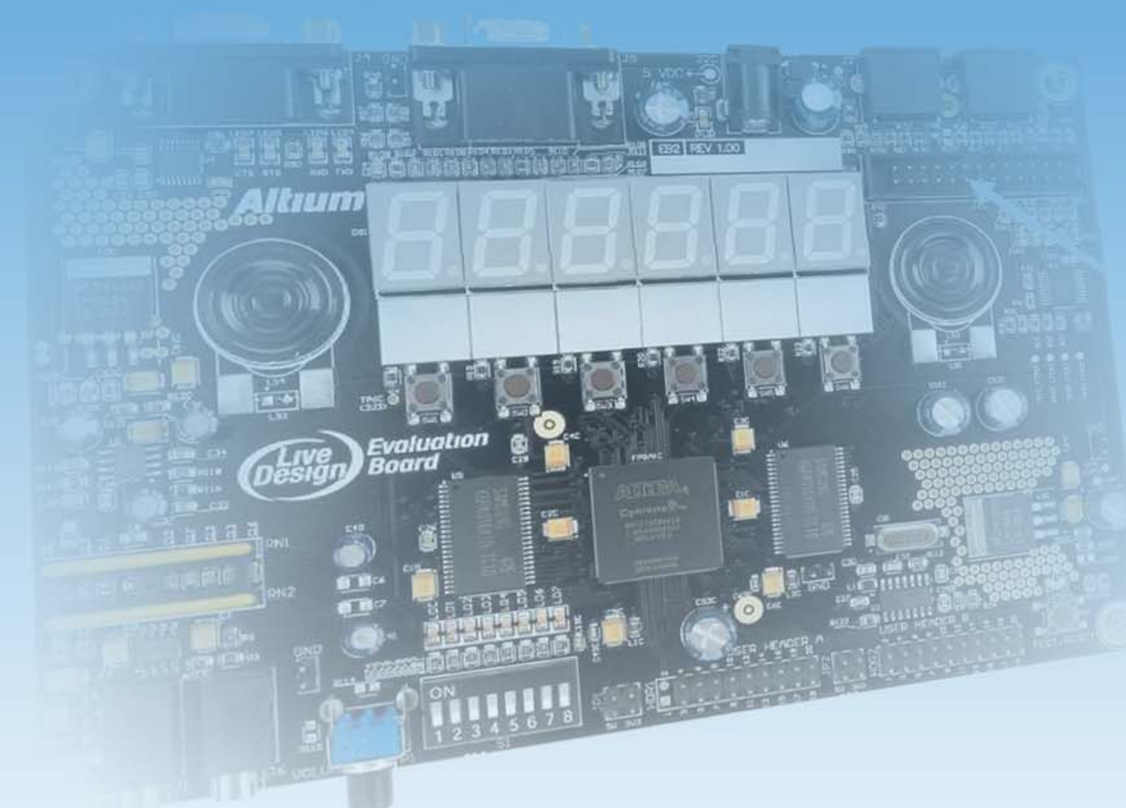
end buton_paraziti;
```

GABRIEL V. IANA ş.a. - PROGRAMAREA CU LIMBAJE DE DESCRIERE HARDWARE

Gabriel V. Iana, Gheorghe Şerban
Laurenţiu Ionescu, Petre Anghelescu

PROGRAMAREA CU LIMBAJE DE DESCRIERE HARDWARE

Aplicaţii în limbajul VHDL



reset	0	Formula				
A	01EA	Binary ...	0028	0055	0082	00AF
B	935F	Binary ...	8F4C	8FE1	9076	

EDITURA UNIVERSITĂŢII DIN PITEŞTI
2009

PROGRAMAREA CU LIMBAJE DE DESCRIERE HARDWARE

Aplicații în limbajul VHDL

Gabriel V. Iana, Gheorghe Şerban
Laurențiu Ionescu, Petre Anghelescu

UNIVERSITATEA DIN PITESTI
2009

CUPRINS

INTRODUCERE

VHDL este un limbaj de descriere hardware. Prin el se descrie comportamentul unor circuite electronice digitale sau sisteme care pot fi implementate fizic. Denumirea de VHDL vine de la VHSIC Hardware Description Language, VHSIC este însăşi o abreviere de la Very High Speed Integrated Circuits. Acest limbaj reprezintă o inițiativă finanțată de către departamentul de apărare al Statelor Unite în anii 1980.

VHDL este destinat pentru sinteza circuitelor, precum şi simularea acestora. Cu toate acestea, deşi codul VHDL este pe deplin simulabil, nu toate construcțiile sunt sintetizabile. O motivație fundamentală de a utiliza limbaje de descriere hardware cum ar fi VHDL sau Verilog (concurentul VHDL-ului) este faptul că sunt standardizate, nu depind de tehnologia circuitelor pe care se implementează codul, sunt portabile şi reutilizabile. Două din aplicațiile principale imediate ale limbajului VHDL se regăsesc în domeniul dispozitivelor logice programabile - Programmable Logic Devices (inclusiv CPLDs-Complex Programmable Logic Devices şi FPGA-Field Programmable Gate Arrays), precum şi în domeniul circuitelor integrate dedicate pe aplicație - ASICs (Application Specific Integrated Circuits). După ce este scris codul VHDL, acesta poate fi sintetizat iar pentru realizarea fizică a circuitului dorit poate fi implementat într-o structură programabilă (de la Altera, Xilinx, Atmel, etc) sau pot fi trimise pentru fabricarea unui circuit de tip ASIC. Este bine cunoscut faptul că, în prezent, multe circuite complexe (microprocesoare, de exemplu) pot fi concepute folosind o astfel de abordare. O altă observație în ceea ce priveşte limbajul VHDL este că, spre deosebire de programele software care sunt executate secvențial, declarațiile sale sunt în mod inerent concomitente (paralele). Din acest motiv, limbajul VHDL este de obicei mai degrabă menționat ca un cod şi nu ca un program.

Cartea se adresează în primul rând studenților din cadrul secțiilor de Electronică, Calculatoare şi Telecomunicații care au în programă proiectarea circuitelor logice utilizând limbaje de descriere hardware. De asemenea, prin bogatul suport practic pe care îl deține, cartea se adresează oricui doreşte să se inițieze dar şi să aprofundeze în domeniul proiectării cu limbaje de descriere hardware.

Lucrarea este structurată pe 7 capitole, urmate de două anexe, prin care se realizează o introducere treptată în complexitatea problematicii, de la

abordarea mediilor de proiectare cu limbaje hardware (mediul Xilinx ISE la capitolul 1) până la realizarea propriilor pachete de funcții şi de declarații care pot fi utilizate în proiectare (capitolul 7). Fiecare capitol începe cu un breviar teoretic, în care sunt prezentate conceptele legate de tematica acelui capitol. Partea principală a capitolelor este constituită de aplicații. Acestea sunt prezentate sub forma unor probleme urmate de soluțiile propuse (uneori sunt prezentate şi mai multe metode de rezolvare). Toate aplicațiile au fost destinate implementării practice pe machete cu circuite FPGA Spartan 3 XC3S400 existente în cadrul laboratoarelor de Calculatoare numerice, din cadrul Facultății de Electronică, Comunicații şi Calculatoare al Universității din Piteşti. Totuşi, dat fiind gradul de generalitate pe care îl oferă limbajul de descriere hardware VHDL (utilizat în această carte), aplicațiile pot fi implementate şi pe alte sisteme cu alte tipuri de circuite reconfigurabile prin schimbarea corespunzătoare a constrângerilor legate de distribuirea pinilor pe capsulă. Fiecare capitol se termină cu un set de exerciții propuse ce au ca scop stimularea inițiativei personale a studenților.

Capitolul I
Configurarea structurilor hardware reprogramabile de tip FPGA cu XILINX ISE

În acest capitol este prezentată metodologia relativă la implementarea programelor scrise prin limbajul VHDL, pe structura reprogramabilă FPGA de tip SPARTAN 3, folosind pachetul software de dezvoltare XILINX ISE. Pe scurt, sunt abordate următoarele etape: introducerea codului VHDL, simularea şi vizualizarea rezultatelor obţinute, sinteza şi implementarea surselor, iar în final programarea structurii reconfigurabile.

1. Breviar teoretic

1.1. Mediul de proiectare Xilinx ISE

Mediul software de proiectare XILINX ISE (Integrated Software Environment) este utilizat la realizarea şi implementarea completă a unui proiect pentru structurile programabile de tip XILINX. Componenta software ISE Project Navigator facilitează crearea proiectului, organizându-l pe următoarele etape:

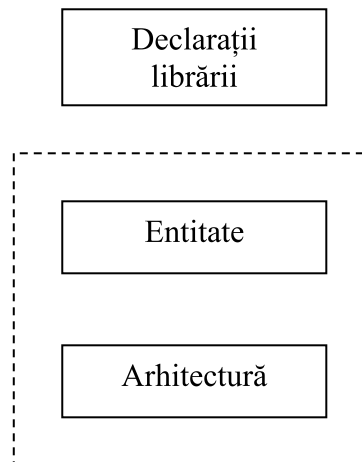
- *Descriere proiect.* Programatorul are posibilitatea să descrie proiectul prin introducerea de coduri sursă HDL (Hardware Description Language) pentru limbajele VHDL, Verilog, Abel sau utilizând schematice şi diagrame cu stări finite;
- *Sinteza.* În cadrul acestei etape fişierele de tip VHDL, Verilog sau schematice sunt transformate în fişiere de tip *netlist* care sunt acceptate ca fişiere de intrare la etapa de implementare;
- *Implementarea.* După sinteză, la implementare, proiectul este adaptat şi transformat din forma logică digitală în forma tehnologică implementabilă pe structura reconfigurabilă aleasă;
- *Verificarea.* Poate fi realizată în toate etapele de implementare ale proiectului. Utilizarea componentelor software de simulare conduce la verificarea completă a funcţionalităţii proiectului sau a unor porţiuni de proiect. De asemenea, pot fi realizate şi verificări directe pe circuit, după programarea acestuia;
- *Configurarea.* După generarea fişierelor de programare (bitstream file) proiectantul are posibilitatea programării circuitului reconfigurabil. În timpul procesului de configurare sunt programate interconexiunile structurii FPGA alese.

În vederea exemplificării modului de implementare a unui modul digital pe o structură de tip FPGA, se prezintă următorul model.

1.2. Structura unui program VHDL

Programul prin care se descrie un modul digital în limbajul VHDL conține trei părți:

- declararea librăriilor care vor fi utilizate în proiect;
- declararea entității modulului ce urmează a fi proiectat;
- descrierea arhitecturii acestuia.



Entitatea descrie interfața modulului digital cu semnalele din mediul exterior. O entitate deja declarată poate fi accesată de către alte entități.

Sintaxă: **entity** nume_entitate **is**
generic (listă_generică);
port (listă_de_porturi);]
end entity nume_entitate;

Prin specificația **entity** se declară numele modulului digital. În plus, pot fi declarați parametrii generici şi porturi care fac parte din această entitate. Porturile declarate într-o entitate sunt vizibile în toate arhitecturile atribuite acesteia. Porturile pot fi de intrare, ieşire, buffer sau bidirecționale (**IN, OUT, BUFFER, INOUT**)

Arhitectura descrie relația dintre intrările şi ieşirile porturilor entității căreia îi este asociată. O arhitectură poate avea asociată doar o singură entitate, dar o entitate poate avea asociate mai multe arhitecturi prin configurație.

Sintaxă:
architecture nume_arhitectură **of** nume_entitate **is**
-- declarații în arhitectură
begin
--specificații_concurente
end [**architecture**] [nume_arhitectură];

Zona declarativă a unei arhitecturi poate conține declarații de tipuri, semnale, constante, subprograme, componente şi grupuri. Specificațiile concurente din corpul arhitecturii definesc legăturile dintre intrările şi ieşirile modulului digital pe care-l reprezintă.

Sintaxa generală a unui program în cod VHDL este următoarea:

```
--Declararea librariilor prin clauza library si use
library nume_librarie;
use nume_librarie.nume_pachet.all;

--Declararea entitatii modulului digital
entity nume_entitate is
        generic(nume generice : type := valori_initiale);
        port(nume_porturi : directie tip port);
end entity nume_entitate; --entity[93]

--Corpul arhitecturii
architecture nume_arhitectura of nume_entitate is
        declaratii arhitectura
begin
        specificatii concurente
        end architecture nume_arhitectura;
```

1.3. Operatori utilizați în limbajul VHDL

Pentru implementarea circuitelor combinaționale, în limbajul VHDL, s-au pus la dispoziția programatorului operatori logici, aritmetici, de comparație, deplasare şi de concatenare.

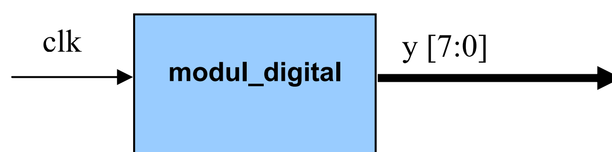
În tabelul de mai jos sunt prezentați, pe scurt, operatorii logici:

Tipul operatorului	Operatori	Tipul datelor
Logic	NOT, AND, NAND, OR, NOR, XOR, XNOR	BIT, BIT_VECTOR, STD_LOGIC, STD_LOGIC_VECTOR STD_UNLOGIC, STD_UNLOGIC_VECTOR
Aritmetic	+, -, *, /, ** (mod, rem, abs)	INTEGER, SIGNED, UNSIGNED
Comparație	=, /=, <, >, <=, >=	aproape toți
Deplasare	sll, srl, sla, sra, rol, ror	BIT_VECTOR
Concatenare	&, (, , ,)	La fel ca la operatorii logici, plus SIGNED şi UNSIGNED

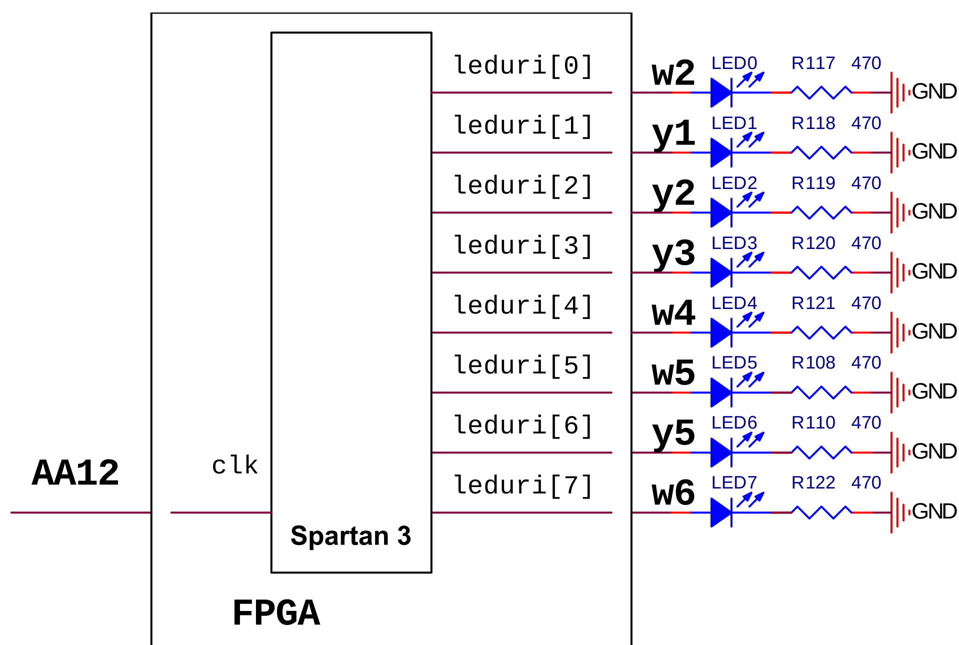
2. Aplicație

Să se descrie etapele de implementare ale unui modul digital numărător binar pe 4 biți, în limbajul VHDL, pe sistemul reconfigurabil cu circuitul FPGA SPARTAN3 din laborator.

Porturile de intrare/ieşire ale modulului digital sunt prezentate în figura următoare:



Verificarea modulului digital va fi realizată folosind schema electrică din figura de mai jos. Această schemă este doar o mică parte din schema electrică a sistemului reconfigurabil de dezvoltare SPARTAN 3:



Notă: Se va consulta schema electrică completă a sistemului de dezvoltare şi se vor identifica pe aceasta elementele de circuit.

Pinii de interconectare ai structurii FPGA cu modulul digital descris în VHDL sunt marcați cu bold (**Y6**, **Y17**, **C6**, **B8**, **E7** şi **C5**).

2.1. Crearea proiectului

Se lansează programul XILINX ISE din mediul Windows.

Primul pas în realizarea modulului digital constă în crearea proiectului.

Crearea unui proiect se face plecând de la fereastra generală, prin selectarea **File -> New Project**.

Se deschide o fereastră de dialog care va fi completată după cum urmează:

New Project

Enter a Name and Location for the Project

Project Name: modul_digital

Project Location: E:\Cursuri Facultate\Structuri hardware re ...

Select the type of Top-Level module for the Project

Top-Level Module Type: HDL

< Back | Next > | Cancel | Help

Project Name – se introduce numele proiectului (modul_digital);
Project Location – se selectează directorul în care se doreşte salvarea proiectului;
Top-Level Module Type – fixat ca fiind de tipul HDL.

În acest caz, proiectul poartă denumirea „modul_digital”.
Se selectează butonul **Next,** după care apare următoarea fereastră:

New Project

Select the Device and Design Flow for the Project

Property Name	Value
Device Family	Spartan3
Device	xc3s400
Package	fg456
Speed Grade	-4
Top-Level Module Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator
Generated Simulation Language	VHDL

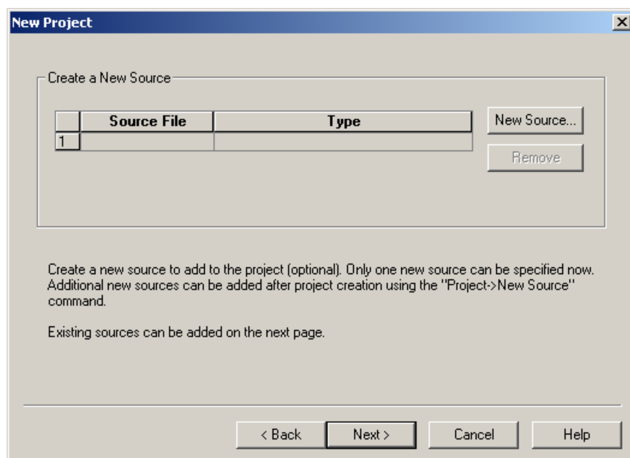
< Back | Next > | Cancel | Help

În vederea completării proprietăților ce apar în fereastră, se selectează linia corespunzătoare acestora din zona **Value** şi se completează din listele afişate următoarele valori:

- **Device Family:** structura reconfigurabilă utilizată în cadrul laboratorului este *Spartan 3*;
- **Device:** va fi introdus codul circuitului utilizat. În acest caz este *xc3s400*;
- **Package:** se indică tipul capsulei şi numărul de pini; circuitul Spartan folosit are un package de tipul *fg456*;
- **Speed Grade:** viteza de propagare a semnalului de ceas în FPGA este *-4*;
- **Syntesis Tool:** *XST[VHDL/Verilog]*;
- **Simulator**: programul ales pentru simularea şi verificarea modulului digital este *ISE Simulator*;
- **Generated Simulator Language:** *VHDL*.

Toate fişierele proiectului vor fi salvate automat într-un subdirector al acestuia. Un proiect poate avea decât un singur fişier HDL, de tip *top-level* (este practic modulul principal, capul ierarhiei, care uneşte toate celelalte submodule). Modulele digitale pot fi adăugate secvențial la proiect pentru a forma o structură ierarhică şi modulară.

Se selectează butonul **Next,** după care apare următoarea fereastră:



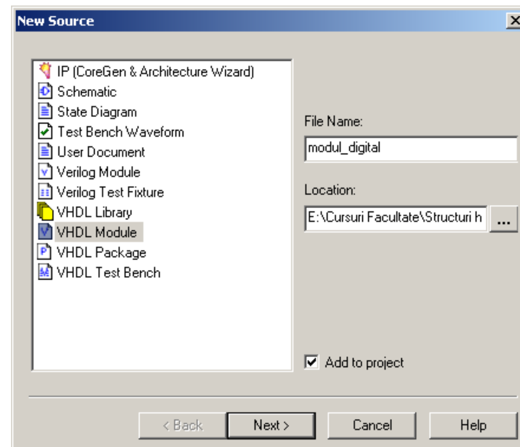
Se creează un fişier sursă nou prin selectarea butonului **New Source**.

2.2. Crearea şi introducerea unui fişier sursă VHDL

În cele ce urmează se va introduce în mediul Xilinx ISE un program în limbajul VHDL. Se creează un fişier VHDL, cu extensia **.vhd* ce va fi scris utilizând editorul programului XILINX.

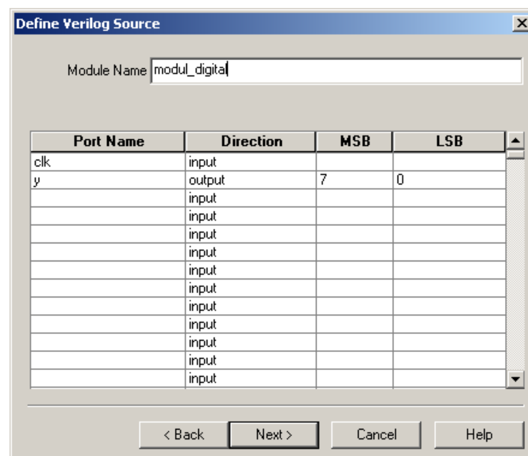
În figura de mai sus se selectează butonul **New Source** pentru a fi introdus un fişier nou. Dacă nu se doreşte acest lucru, ci doar adăugarea unor fişiere deja existente, se selectează butonul **Next**.

Va fi afişat ecranul:



După cum se poate vedea în figura de mai sus, pot fi create mai multe tipuri de fişiere ce pot fi adăugate la proiectul nostru.

În acest caz, se va selecta un fişier de tip **VHDL Module**, iar la zona de editare **File Name** se va introduce „modul_digital". Opţiunea **Add to project** trebuie selectată permanent pentru ca fişierul creat sa fie ataşat la proiect. În continuare se selectează butonul **Next**.



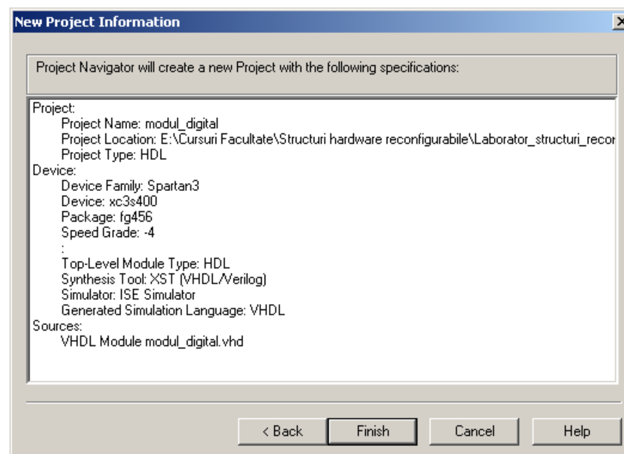
În coloana cu numele **Port Name** sunt introduse numele tuturor pinilor cu direcţia specificată în coloana **Direction** (direcţia poate fi *input*, *output*, *inout*). În cazul în care porturile sunt vectori sau magistrale, pe coloanele MSB şi LSB se specifică dimensiunea acestora.

Porturile introduse sunt următoarele:

clk *input*
y *output* *7 .. 0*

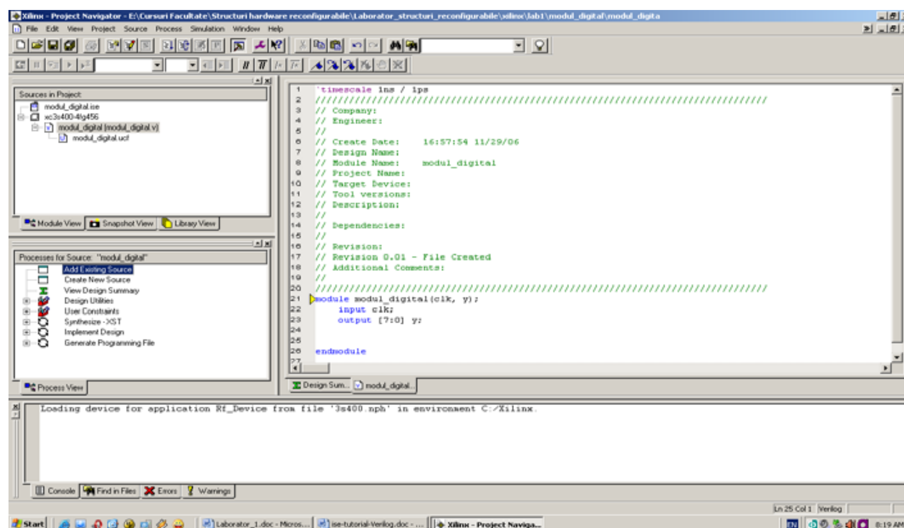
După completarea tuturor porturilor modulului digital, se selectează butonul **Next** şi apare fereastra **New Source Information,** în care sunt afişate toate datele completate până în prezent despre proiectul ce urmează a fi implementat.

Se selectează butonul **Finish**. Se revine în fereastra anterioară, dar care are adăugat fişierul deja creat. În continuare se selectează butonul **Next**. Va apare o fereastră ce dă posibilitatea utilizatorului să adauge proiecte noi. În final, după o nouă selectare a butonului **Next** va fi afişată fereastra de mai jos, în care sunt prezentate toate specificațiile proiectului creat.



Se selectează butonul **Finish**, iar fişierul VHDL va apare în zona **Sources in Project** din **Project Navigator**.

2.3. Editarea fişierului VHDL



Fişierul sursă poate fi vizualizat în fereastra **Project Navigator**. Fereastra în care este afişat fişierul sursă poate fi utilizată ca un editor în vederea completării sau modificării programului VHDL. Este recomandat ca programul să fie salvat periodic prin comanda **File → Save** din meniul principal. Programele VHDL pot fi editate în orice editor de text şi ataşate la proiect prin comanda **Add Copy Source**.

Programul VHDL va fi completat cu liniile următoare de cod sursă:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modul_digital is
   Port ( clk : in std_logic;
         y : out std_logic_vector(7 downto 0));
end modul_digital;

architecture Behavioral of modul_digital is
          signal temporar: std_logic_vector(2 downto 0):= (others => '0');
begin
         process(clk)
         begin
          if (clk'event and clk = '1')          then
                  temporar <= temporar + '1';
          end if;
         end process;

         process (temporar)
         begin
                  case (temporar) is
                           when "000" =>  y <= "10000001";
                           when "001" =>  y <= "01000010";
                           when "010" =>  y <= "00100100";
                           when "011" =>  y <= "00011000";
                           when "100" =>  y <= "00100100";
                           when "101" =>  y <= "01000010";
                           when "110" =>  y <= "10000001";
                           when others => y <= (others => '1');
                  end case;
         end process;
end Behavioral;
```

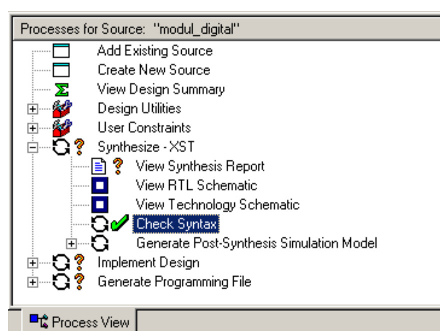
Programul este format din doua blocuri. Primul bloc are funcția de numărător pe 3 biți, iar cel de-al doilea este un decodor binar într-o formă definită de proiectant.

Legătura dintre blocuri este realizată prin doi regiştri declaraţi în prima parte a modulului digital (*y* şi *temporar*). Regiştrii au avantajul că pot păstra valorile salvate pe o perioadă nedeterminată de timp.

În vederea determinării corectitudinii codului sursă din fişier se va verifica sintaxa acestuia.

În cadrul ferestrei **Process View** se selectează **Check Syntax**. După verificare, dacă nu există nicio eroare va apare semnul ✔ în dreptul acesteia.

În caz contrar, erorile vor fi afişate în partea de jos a display-ului, în fereastra **Console**.



2.4. Simularea proiectului

Simularea proiectului este necesară pentru verificarea corectitudinii funcţionale ale modulelor digitale create. În prima fază, simularea se face prin vizualizarea formelor de undă ale stimulilor de intrare şi ale semnalelor rezultate la ieşirile modulelor digitale. În acest scop este necesară crearea unui fişier de tip **Test Bench Waveform**, urmărind paşii de mai jos:

- Se selectează fişierul sursă *modul_digital.vhd* din fereastra **Sources in Project**;
- Se creează o sursă nouă selectând **Project → New Source**;
- În fereastra **New Source Window** se selectează fişierul de tip **Test Bench Waveform** şi i se alocă numele *fisier_test*;
- Se selectează butonul **Next** după care apare o fereastră **Select** în care vor apare fişierele sursă editate de utilizator până în prezent. În cazul nostru apare doar fişierul *modul_digital*;
- Se selectează butonul **Next** şi apoi butonul **Finish**.

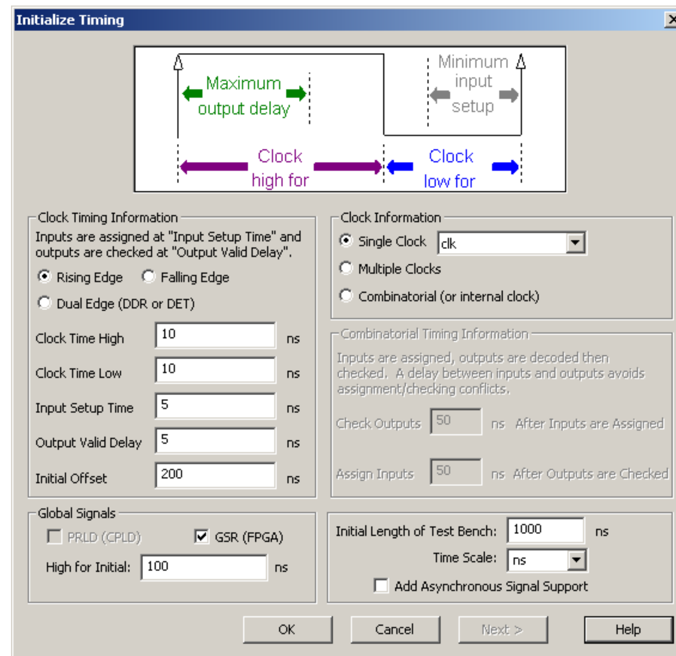
În final, apare fereastra **Initialize timing** şi se fac următoarele setări:

- în zona **Clock Timing Information** se selectează frontul de ceas pozitiv **Rising Edge**, **Clock Time High** de 10ns, **Clock Time Low** de 10ns, **Input Setup Time** de 5ns, **Output setup time** de 5ns, iar **Initial Offset** de 200ns;

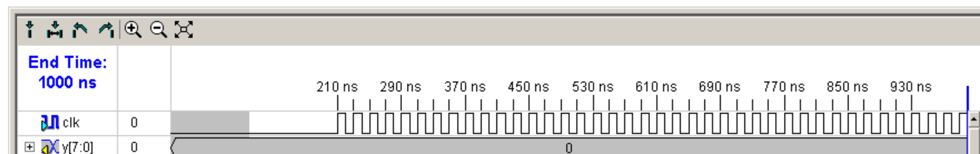
- în zona **Global Signals** se selectează **GSR(FPGA)** şi semnalul de ceas, care iniţial, va fi 1 logic pentru 100ns;

- în zona **Clock Information**, se selectează un singur semnal de ceas **Single Clock** şi se introduce denumirea acestuia din programul VHDL (în cazul nostru semnalul **clk)**;

- în final, timpul de simulare **Initial Length Test Bench** va fi setat la valoarea de 1000ns.

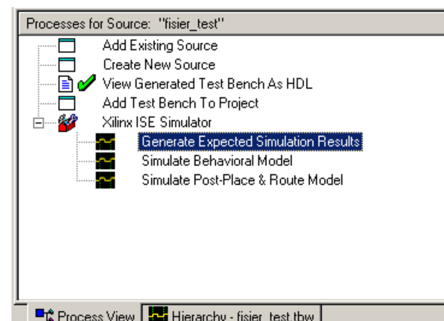


Pentru finalizarea completării acestui tabel se selectează butonul **OK**, după care apare în **Project Navigator** fereastra ce cuprinde semnele de intrare, respectiv de ieşire cu formele respective de undă.



Această fereastră va fi salvată prin comanda **File** → **Save** din meniu. În fereastra **Sources in Project** va apare un fişier cu numele *fisier_test.tbw*.

După selectarea acestui fişier, fereastra **Process View** va avea conţinutul din figura alăturată.



Prin selectarea procesului **Generate Expected Simulation Results** se realizează simularea funcţională în care nu se ţine cont de timpii de propagare prin circuit.

În acest pas vor fi afişate formele de undă ale semnalelor de ieşire, funcţie de cele ale semnalelor de intrare. Practic, fereastra de afişare prezintă funcţia pe care programul urmează să o realizeze.

După terminarea procesului de generare a rezultatelor simulării, va fi afişat următorul grafic:

Vizualizarea la nivel de bit a semnalului de ieşire se realizează prin selectarea semnului + din dreptul acestuia.

În mod similar se generează simularea comportamentală (**Simulate Behavioral Model**) şi simularea după post procesare (**Simulate Port-Place & Route Model**).

În vederea realizării modelului final al modulului digital vor fi adăugate următoarele linii de cod VHDL:

- în zona declarativă a arhitecturii se adaugă linia signal semaf :std_logic;
- se adaugă următoarele procese:

```
process(clk)
        variable cont_temp:std_logic_vector(23 downto 0);
begin
        if (clk'event and clk = '1')        then
                semaf <= '0';
                cont_temp := cont_temp+1;
                if (cont_temp = x"FAF080") then
                        cont_temp := x"000000";
                        semaf <= '1';
                end if;
        end if;
end process;

process(clk)
begin
        if (clk'event and clk = '1')        then
                if semaf = '1'     then
                        temporar <= temporar + '1';
                end if;
        end if;
end process;
```

Programul final este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modul_digital is
   Port ( clk : in std_logic;
          y : out std_logic_vector(7 downto 0));
end modul_digital;

architecture Behavioral of modul_digital is
             signal temporar:          std_logic_vector(2 downto 0):= (others => '0');
             signal semaf :std_logic;
begin
         process(clk)
                 variable cont_temp:std_logic_vector(23 downto 0);
         begin
                 if (clk'event and clk = '1')           then
                         semaf <= '0';
                         cont_temp := cont_temp+1;
                         if (cont_temp = x"FAF080") then
                                 cont_temp := x"000000";
                                 semaf <= '1';
                         end if;
                 end if;
         end process;
         process(clk)
         begin
                 if (clk'event and clk = '1')           then
                         if semaf = '1'     then     temporar <= temporar + '1';
                         end if;
                 end if;
         end process;
         process (temporar)
         begin
                 case (temporar) is
                         when "000" => y <= "10000001";
                         when "001" => y <= "01000010";
                         when "010" => y <= "00100100";
                         when "011" => y <= "00011000";
                         when "100" => y <= "00100100";
                         when "101" => y <= "01000010";
                         when "110" => y <= "10000001";
                         when others => y <= (others => '1');
                 end case;
         end process;
end Behavioral;
```

2.5. Crearea şi editarea constrângerilor

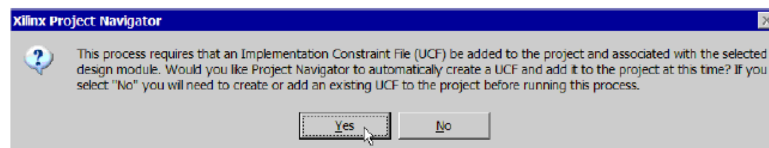
În orice proiect este necesară utilizarea constrângerilor în vederea respectării cerinţelor referitoare la timpii de propagare, aria ocupată şi atribuirea pinilor. În acest scop sunt posibile trei categorii de constrângeri.

2.5.1. Constrângeri referitoare la domeniul timp

Minimal, fiecărui proiect trebuie să i se specifice constrângeri referitoare la perioada semnalului de ceas şi a timpului de offset, în vederea respectării vitezei de lucru a circuitului. Paşii de realizare a unui fişier de constrângeri sunt următorii:

- se selectează fişierul sursă *modul_digital* din fereastra **Sources in Project**; în fereastra **Process for Source** vor apare procesele corespunzătoare;
- se expandează grupul **User Constraints** în fereastra **Process for Source** şi se selectează procesul **Create Timing Constraints**.

Mediul de dezvoltare Xilinx ISE rulează automat procesul de sinteză şi creează un fişier de tip UCF (User Constraints File). După apariţia mesajului

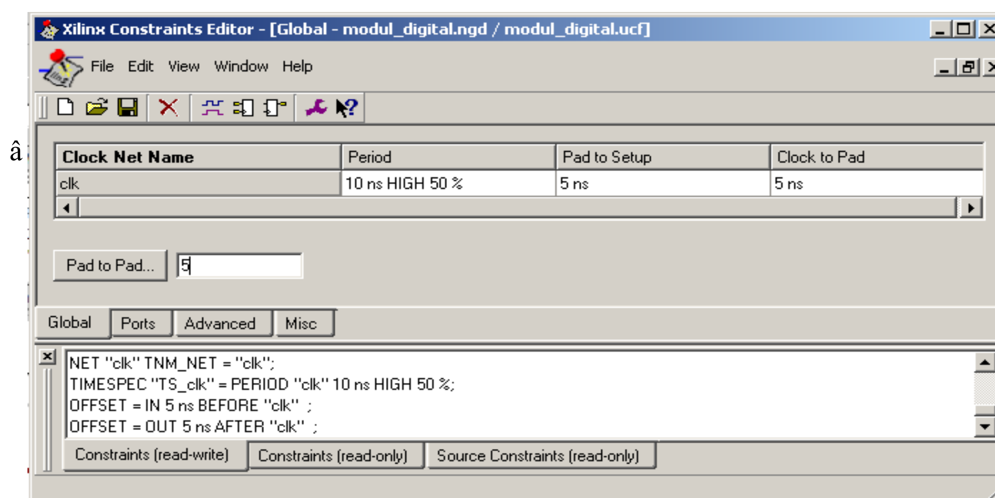


se va selecta butonul **Yes,** fapt ce permite ataşarea fişierului UCF la proiect, iar în final, afişarea editorului de constrângeri.

În fereastra **Global** a editorului de constrângeri se vor introduce următoarele valori:

Period - 20ns, **Pad to Setup** - 10ns şi **Clock to Pad** - 10ns.

â



Se salvează fişierul prin comanda **File** → **Save** după care se închide editorul de constrângeri.

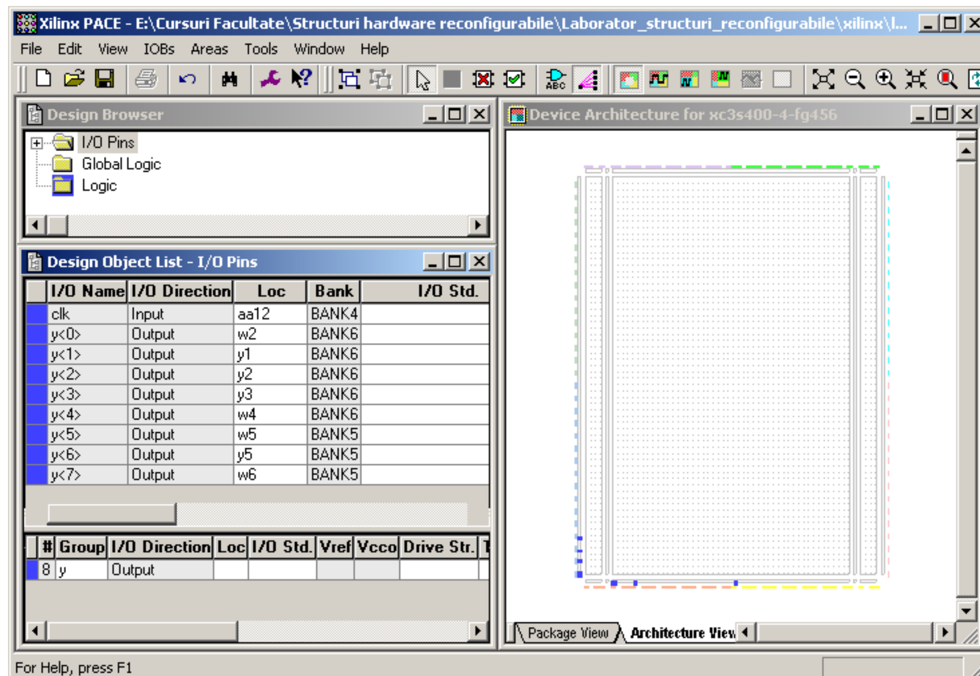
2.5.2. Constrângeri referitoare la atribuirea pinilor

Prin aceste constrângeri sunt realizate asocierile dintre porturile modulului digital declarate în entitate şi pinii circuitului de tip FPGA utilizat.

În cazul nostru vor fi introduse următoarele constrângeri:

clk → aa12
y[0] → w2
y[1] → y1
y[2] → y2
y[3] → y3
y[4] → w4
y[5] → w5
y[6] → y5
y[7] → w6

Crearea unui astfel de fişier se face prin selectarea procesului **Assign Package Pins** în grupul **User Constraints**. Programul **Xilinx Pinout and Area Constraints Editor (PACE)** se va deschide automat.



După completarea constrângerilor se selectează **File → Save**. La apariţia casetei de atenţionare se selectează **XST Default < >** şi apoi butonul **OK**. În final se închide editorul.

2.6. Sinteza proiectului

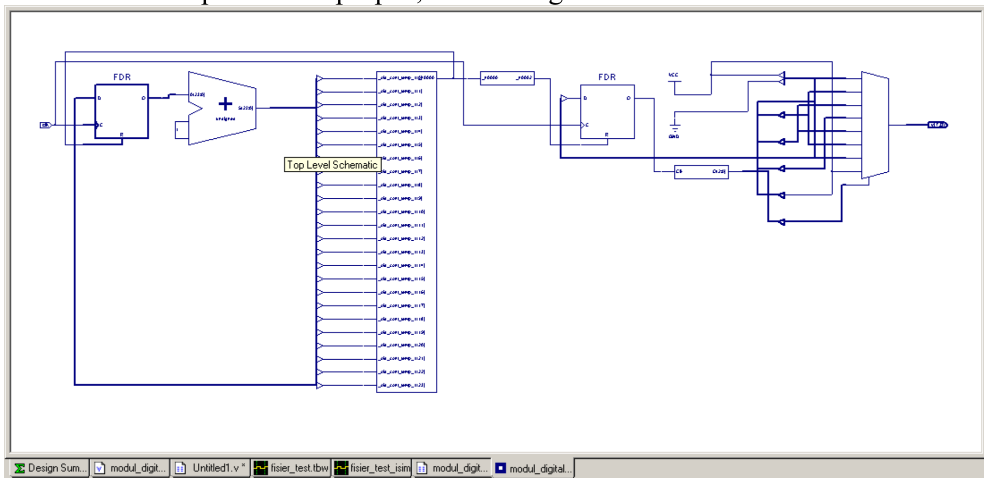
După ce proiectul a fost completat, verificat şi s-au adăugat fişierele de constrângeri, acesta poate fi sintetizat şi apoi implementat pentru a se face, în final, programarea structurii reconfigurabile.

Sinteza se realizează prin selectarea (dublu clic) a grupului **Synthesize – XST**.

Dacă procesul nu a generat erori poate fi vizualizat primul raport referitor la modulul digital. În acest scop, se selectează grupul **Synthesize – XST,** după care se expandează şi se alege **View Synthesis Report**. În cadrul raportului, printre alte informaţii, se regăsesc specificaţii referitoare la gradul de ocupare al ariei şi a timpului de propagare prin structura reconfigurabilă.

În cadrul aceluiaşi grup, dacă se selectează **View RTL Synthesis**, se vizualizează schema logică generată după sinteza modulului digital.

În cazul proiectului propus, schema logică este următoarea:



Selectarea (dublu clic) fiecărui bloc poate conduce la o descriere mai detaliată la nivel de schemă.

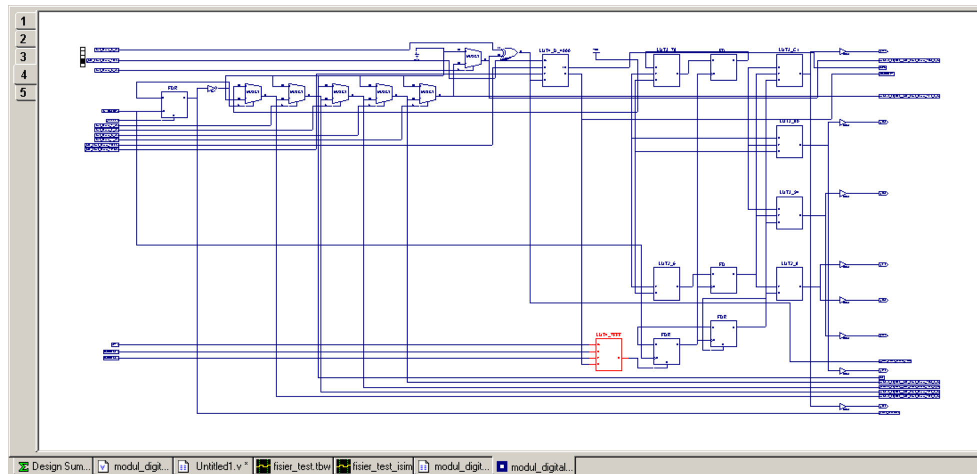
Tot în grupul **Synthesize – XST,** dacă se selectează **View Technology Schematic** se vizualizează schema tehnologică a modulului digital.

Practic structura digitală este formată din blocuri logice configurabile predefinite. Acestea, la rândul lor, sunt formate din SLICE-uri, un SLICE având în componenţă două generatoare de semnal.

În final, un generator de semnal este format dintr-un LUT (Look Up Table), memorie, multiplexoare şi bistabili. LUT-ul este elementul principal în

realizarea funcțiilor logice combinaționale. LUT-ul pot fi comparat cu un codor ce are mai multe intrări, respectiv o ieşire la care se obține o anumită funcție logică.

O mică parte din schema tehnologică a modului digital este prezentată în figura de mai jos:



Dacă este selectat (dublu clic) un circuit LUT, se poate vizualiza schema logică pe care o implementează şi tabela de adevăr corespunzătoare.

2.7. Implementarea proiectului

Implementarea constă în următoarele procese: **Translate, Map** şi **Place & Route**.

Translatarea serveşte la trecerea de la schema logică rezultată în urma sintezei la primitive Xilinx.

Map-area este procesul în care circuitul este descris la nivel de componente fizice în structura FPGA:

Procesul de **Plasare şi rutare** constă în formarea fişierelor ce specifică rutările dintre componentele interne ale structurii reconfigurabile, în vederea generării fişierelor de configurare.

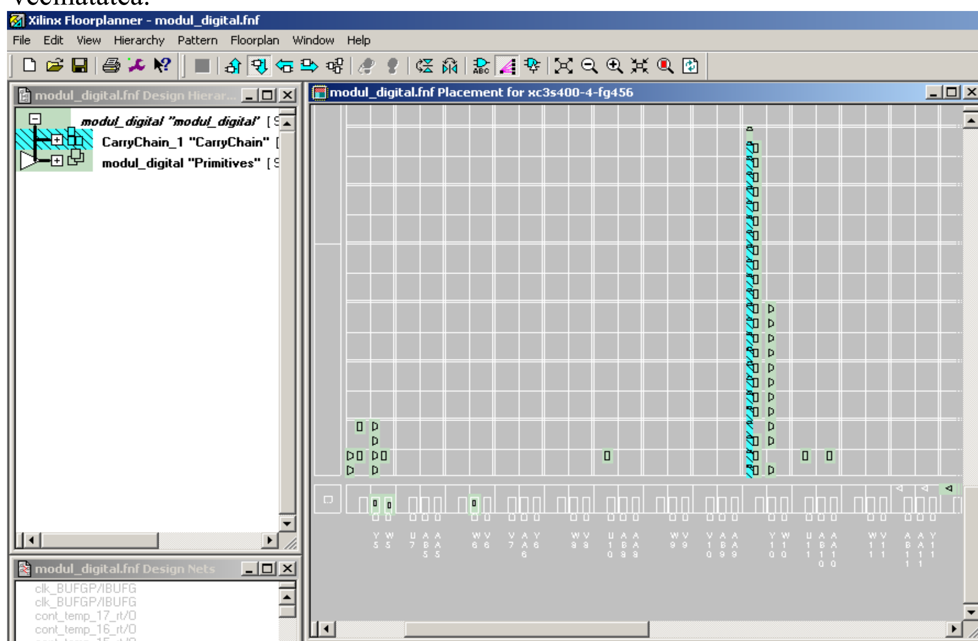
Procesul de implementare se realizează prin selectarea (dublu clic) a grupului **Implement Design**. Dacă procesul s-a terminat fără erori se poate trece la etapa de generare a fişierelor de configurare.

2.8. Verificarea procesului de implementare

Programul **Floorplanner** este utilizat pentru verificarea pinilor şi plasarea acestora. În plus, mai este folosit pentru verificarea utilizării corecte a grupurilor de celule folosite în modulul digital ce se doreşte a fi creat.

În acest scop, se selectează fişierul *modul_digital*. Se expandează grupul **Place&Route** şi se selectează (dublu clic) procesul **View/Edit Placed Design (Floorplanner)**.

Selectarea porţiunii de circuit utilizată se realizează prin comanda **View Zoom → Toolbox**. În figura de mai jos se observă primitivele utilizate în proiectul propus. Dacă se selectează una dintre acestea, apar automat interconexiunile cu vecinătatea.



În fereastra **Processes for Source** se selectează procesul **View Design Summary** şi deschide următoarea fereastră, în care se specifică gradul de ocupare al structurii reconfigurabile.

Design Overview for modul_digital

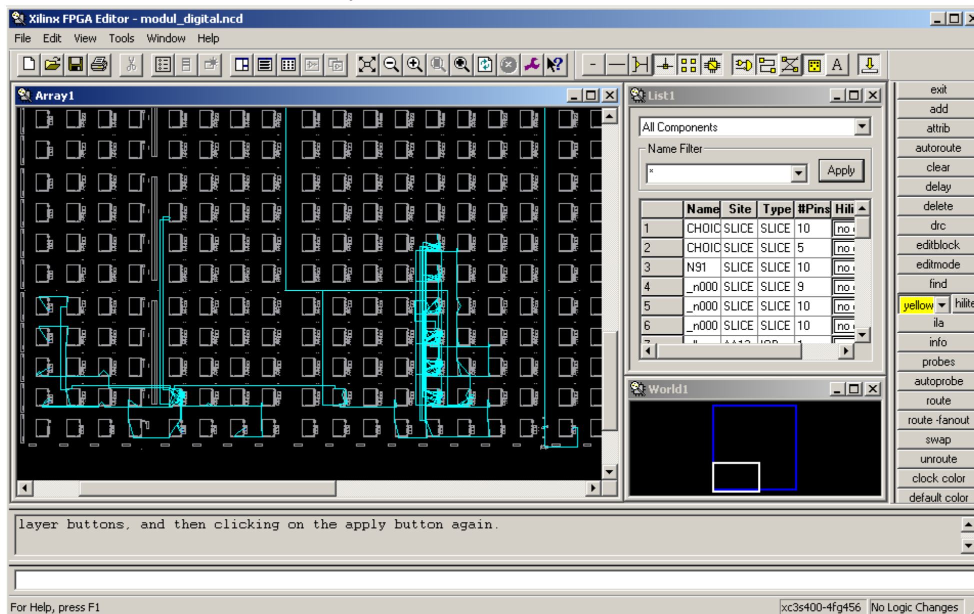
Property	Value
Project Name:	e:\cursuri facultate\structuri hardware reconfigurabile\laborator_structuri_reconfigurabile\xilinx\lab1\modul_digital
Target Device:	xc3s400
Constraints File:	modul_digital.ucf
Report Generated:	Thursday 11/30/06 at 15:25
Printable Summary (View as HTML)	modul_digital_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	28	7,168	1%	
Number of 4 input LUTs:	18	7,168	1%	
Logic Distribution:				
Number of occupied Slices:	25	3,584	1%	
Number of Slices containing only related logic:	25	25	100%	
Number of Slices containing unrelated logic:	0	25	0%	
Total Number 4 input LUTs:	41	7,168	1%	
Number used as logic:	18			
Number used as a route-thru:	23			

În mod asemănător se selectează din grupul **Place & Route** procesul **View/Edit Routed Design (FPGA)**. Se selectează icoana pentru a fi

evidențiate semnalele de rutare din interiorul structurii reconfigurabile. Pentru o mai bună vizualizare se mărește zona de interes.



Dacă se intră suficient de mult în detaliu se pot vedea rutările în interiorul CLB-urilor şi a SLICE-urilor.

2.10. Configurarea circuitului FPGA

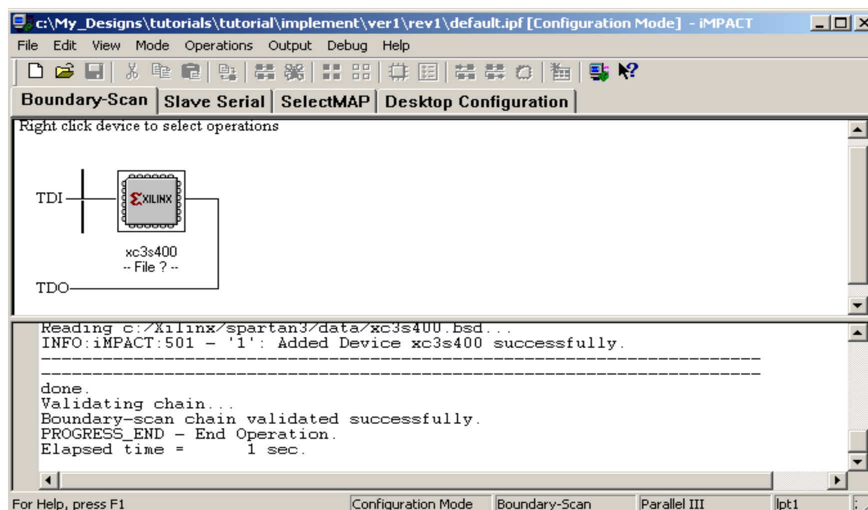
În final, pentru generarea şi implementarea fizică a fişierului binar în structura hardware reconfigurabilă, se apelează programul **IMPACT**.

În cadrul grupului **Generate Programming File** din fereastra **Processes for Source** se selectează procesul **Configure Device (iMPACT)**.

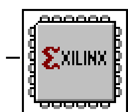
În caseta de dialog **Configure Devices** se selectează opțiunea **Boundary-Scan Mode** şi apoi butonul **Next**. Dacă apare mesajul identificării circuitului se selectează butonul **OK**.

Se continuă prin apariția ferestrei de dialog **Assign New Configuration File**, în care se selectează fişierul binar corespunzător proiectului realizat, iar mesajul de atenționare ce poate să apară nu va fi luat în considerare.

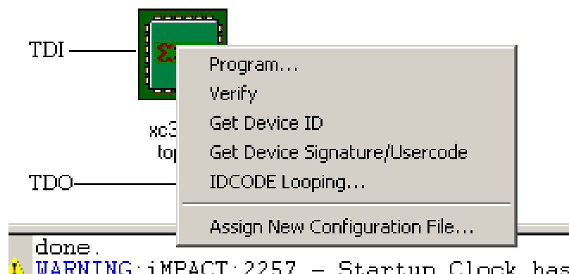
În final, apare fereastra următoare, prin care se poate realiza programarea circuitului.



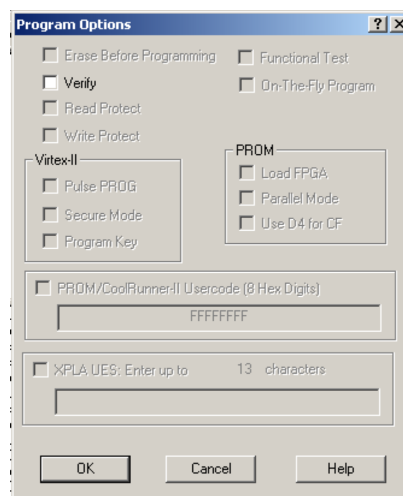
Pentru a fi introdus fişierul binar de configurare a structurii reconfigurabile se selectează pictograma:



Se ignoră mesajul de atenţionare, după care se programează circuitul FPGA prin selectarea câmpului **Program…** din figura alăturată.



Urmează apariţia fereastrei **Program Opţions**:

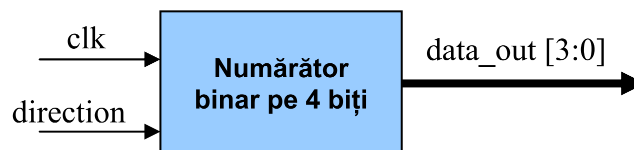


După selectarea butonului OK se porneşte procesul de descărcare a fişierului de configurare în circuit.

La terminarea descărcării, circuitul se configurează automat, iar rezultatele implementării proiectului apar la nivelul platformei de dezvoltare.

3. Exerciții

1. Se citesc informațiile prezentate anterior şi se exemplifică pe calculatoarele existente;
2. Se va realiza şi implementa un numărător binar reversibil pe 4 biți cu porturile de intrare/ieşire prezentate în figura de mai jos:



Se va utiliza următorul cod sursă VHDL al modulului digital:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

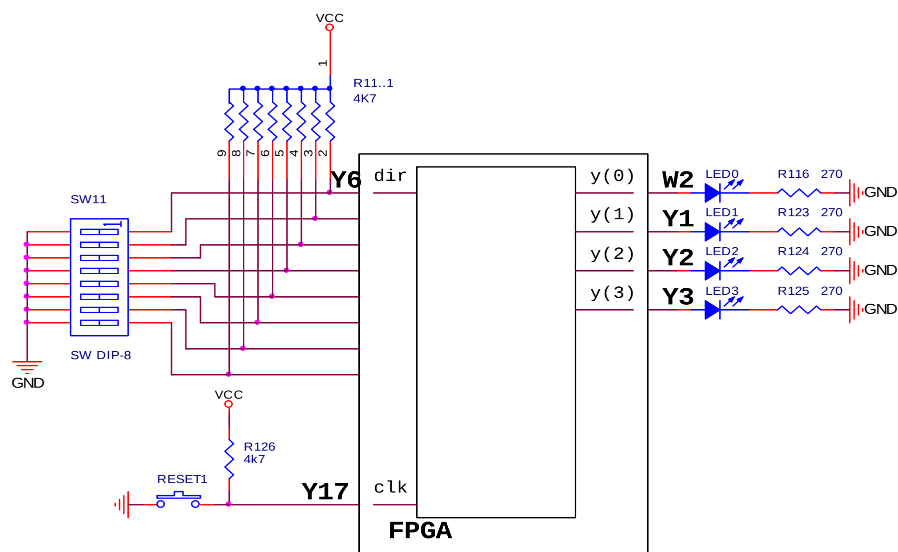
entity counter_UpDown is
   Port ( clk : in std_logic;
          direction : in std_logic;
          data_out : out std_logic_vector(3 downto 0));
end counter_UpDown;

architecture Behavioral of counter_UpDown is
begin
        process (clk)
                variable count_int: std_logic_vector(3 downto 0):= "0000";
        begin
                if (clk'event and clk = '1') then
                        if (direction = '0') then
                                count_int := count_int + '1';
                else
                                count_int := count_int - '1';
                        end if;
                end if;
                data_out <= count_int;
        end process;
end Behavioral;
```

Se vor parcurge următorii paşi:
- crearea unui proiect nou;
- inserarea programului de mai sus ca sursă VHDL;
- simularea modulului digital;

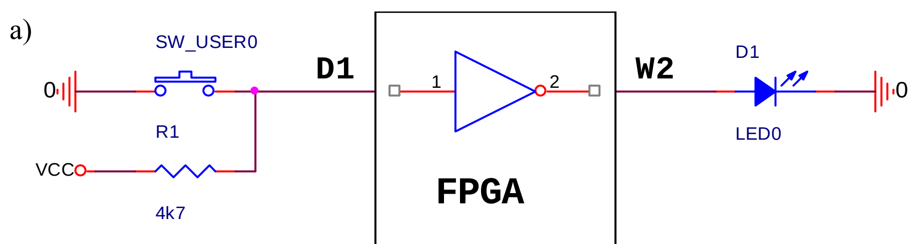
- crearea fişierului de constrângeri al pinilor după schema de mai jos (fără a mai utiliza constrângeri de timp);
- realizarea sintezei acestuia;
- vizualizarea schemelor logice, respectiv tehnologice;
- simularea modelului comportamental rezultat după sinteză;
- implementarea proiectului;
- vizualizarea rulării structurii fizice;
- simularea Post-Place & Post Route a modelului;
- configurarea circuitului fizic.

Acest modul digital va fi inserat în următoarea schemă electrică:

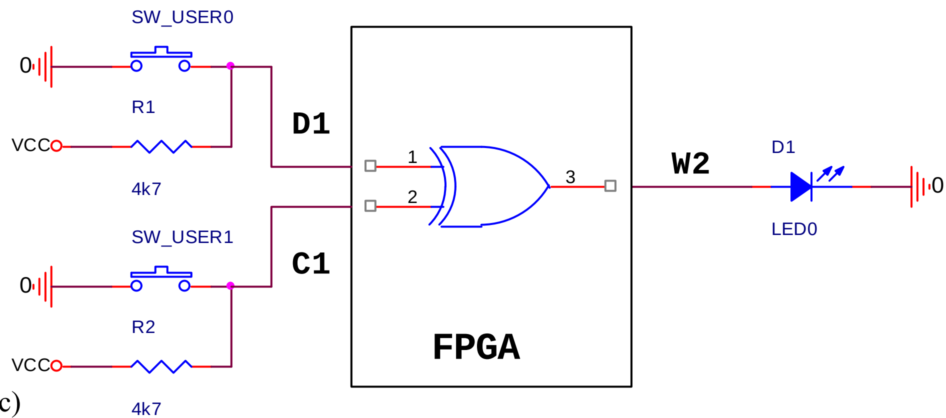


3. Respectând modelul de la punctul anterior, să se realizeze implementarea hardware a funcţiilor logice corespunzătoare pentru funcţionarea următoarelor scheme:

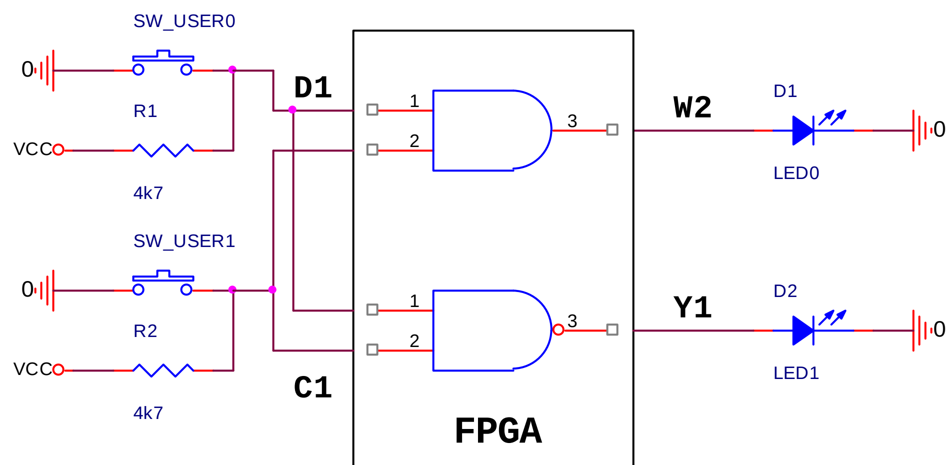
a)



b)



c)



4. După modelul de la punctul 2 să se implementeze hardware ecuațiile booleene ale următoarelor tabele de adevăr, în forma directă şi forma canonică.

a) Tabela de adevăr pentru funcția care realizează codarea unui numărător binar pe 2 biți:

a[0]	a[1]	y[0]	y[1]	y[2]	y[3]
0	0	0	0	0	1
0	1	0	0	1	1
1	0	0	1	1	1
1	1	1	1	1	1

a[0] SW USER0

a[1] SW USER1

y[0] LED 0

y[1] LED 1

y[2] LED 2

y[3] LED 3

b). Tabela de adevăr pentru funcția care verifică paritatea impară a unui număr binar pe 4 biți:

a[0]	a[1]	a[2]	a[3]	y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

a[0] SW USER0

a[1] SW USER1

a[2] SW USER2

a[3] SW USER3

y[0] LED 0

Se va realiza corespondența dintre porturile entităților și pinilor circuitului programabil, după configurațiile din partea dreaptă a tabelelor de adevăr de mai sus.

Capitolul 2
Programarea structurilor hardware reprogramabile prin descrieri concurente

În acest capitol sunt prezentate implementări cu HDL utilizând descrieri cu specificații concurente. Aplicațiile propuse cuprind o parte din specificațiile concurente ce pot fi sintetizate şi implementate pe sistemul reconfigurabil din laborator, utilizând descrieri de tip flux de date sau de tip structural.

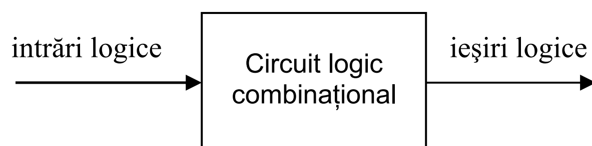
1. Breviar teoretic

1.1. Descrierea cu specificații concurente

Prin intermediul limbajelor de descriere hardware pot fi proiectate module digitale independente, interconectate între ele prin semnale şi care funcționează în paralel.

Limbajul de descriere hardware prezintă mecanisme de descriere paralelă cu specificații concurente a modulelor digitale combinaționale.

Logica combinațională este o structură digitală în care ieşirile depind numai de intrări.



În cadrul unui program scris în limbajul VHDL, zona de descriere concurentă se regăseşte între specificațiile **begin** şi **end** ale unei arhitecturi.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
-- alte incluziuni de librării şi pachete;

entity modul_digital is
            generic ( --declarații de constante generice )
             port(
```

```
                - - -
            );
end modul_digital;


architecture descriere of modul_digital is
begin

        - - - zona de specificații concurente

end descriere;
```

1.2. Atribuirea condițională a semnalelor

Atribuirea semnalelor se face în interiorul arhitecturii sau a proceselor.
Atribuirea simplă a unui semnal este realizată prin operatorul „<=”.
Atribuirea condițională în domeniul concurent se realizează prin specificația WHEN/ELSE.
Sintaxă:

```
LABEL1:     -- etichetă optională
        SIG_NAME <= <expresie> when <condiție> else
                     ---
                    <expresie> when <condiție> else
                    <expresie>;
```

Modificarea valorii logice a unui semnal se face numai dacă este îndeplinită o anumită condiție booleană. Altfel, este luată în considerare condiția următoare care apare după clauza ELSE. Întotdeauna, o atribuire condițională trebuie să se termine cu specificația ELSE.

1.3. Atribuirea selectivă a semnalelor

Atribuirea selectivă a semnalelor este realizată cu specificația WITH/SELECT. În acest caz, spre deosebire de atribuirea condițională, trebuie incluse toate combinațiile posibile în declarație.

```
LABEL1:     -- etichetă opțională
        with <expresie de selecție> select
                SIG_NAME <= <expresie> when <selectie>,
                            <expresie> when <selectie>,
                             ---
                            <expresie> when others;
```

Pentru eliminarea tuturor posibilităților de selecție din expresia condițională, la sfârşitul specificației de atribuire este obligatorie introducerea clauzei WHEN OTHERS.

1.4. Descrierea structurală

Componenta reprezintă o pereche entitate/arhitectură şi specifică un susbsistem care poate fi introdus şi interconectat (instanțiat) în altă arhitectură pe o metodologie ierarhică. Pentru a fi utilizată, componenta este *declarată* în zona declarativă a arhitecturii, după care inserarea acesteia în alte module se realizează prin *instanțiere*.

Declararea unei componente

Declarația unei componente reprezintă primul pas în procesul de utilizare al acesteia într-un modul digital.

Sintaxa:

```
component component_name [ is ]
   generic (generic_list);
   port (port_list);
end component component_name;
```

Declarația componentei (sintaxa de mai sus) defineşte interfața virtuală (soclul în care va fi introdus circuitul), dar nu indică direct componenta.

O componentă poate fi definită în package-uri, entități, arhitecturi sau declarații de blocuri. În cazul în care, componenta este declarată într-o arhitectură, aceasta trebuie să fie plasată în zona declarativă înainte de clauza **begin**.

Instanțierea unei componente

Al doilea pas în utilizarea componentei constă în instanțierea acesteia, adică realizarea asocierilor de semnale şi atribuirii de valori generice specifice, în cadrul arhitecturii modulului digital.

Sintaxă:

```
eticheta : [ component ] nume_componenta
generic map ( lista_valori_generice )
port map ( lista_porturi );

eticheta : entity nume_entitate [(identificator_arhitectura)]
generic map ( lista_valori_generice )
port map ( lista_porturi );
```

```
eticheta : configuration nume_configuratie
generic map ( lista_valori_generice )
port map ( lista_porturi );
```

Instanțierea unei componente permite păstrarea referințelor unității instanțiate şi valorile actuale ale genericelor, respectiv porturilor acesteia.

Numele componentei instanțiate trebuie să fie acelaşi cu numele componentei declarate. Lista de asociere poate fi realizată după nume sau poziționarea porturilor.

În *asocierea pozițională*, parametrii actuali sunt conectați în aceeaşi ordine cu cea a porturilor în care a fost declarată componenta.

```
U1: poarta PORT MAP(a, b, c);
```

Asocierea după nume dă posibilitatea porturilor şi valorilor generice să fie puse într-o ordine diferită de cea declarată în componentă. Asocierea porturilor sau valorilor generice se face prin operatorul „=>".

```
U1: poarta PORT MAP(in1 =>a, in2 => b,iesire => c);
```

Specificația GENERATE

Specificația GENERATE reprezintă o facilitate furnizată de VHDL pentru realizarea iterativă sau condițională a unor porțiuni de program.

Sintaxă:

```
eticheta: for parametru in interval
generate
[ { declaratii }
begin ]
   { specificatii concurente }
end generate [ eticheta ] ;
```

```
eticheta: if conditie generate
[ { declaratii }
begin ]
   { specificatii concurente }
end generate [ eticheta ] ;
```

Specificația de tip **generate** este utilizată pentru simplificarea descrierii unor porțiuni de program repetitive. De obicei, este utilizată pentru interconectarea unui grup de componente identice folosind o singură componentă care este multiplicată.

O specificație **generate** constă în:

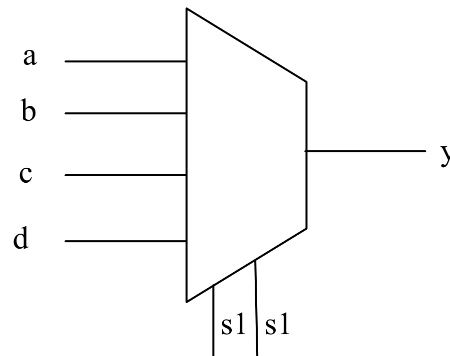
- generarea de scheme (for generate sau if generate);
- parte declarativă (declarații locale de subprograme, tipuri, semnale, constante, componente, atribute, configurații, fişiere şi grupuri);
- specificații concurente.

Elementele limbajului VHDL prezentate în partea teoretică a acestei lucrări de laborator vor fi regăsite în aplicațiile următoare.

2. Aplicații

a) Descrieți în limbajul VHDL un multiplexor de tipul 4:1 cu ajutorul specificației condiționale concurente WHEN/ELSE.

Soluție



Programul VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_a is
   Port ( a,b,c,d : in std_logic;
         s1,s2 : in std_logic;
         y : out std_logic);
end mux_a;

architecture Behavioral of mux_a is
         signal s_temp : std_logic_vector(1 downto 0);
begin
         s_temp <= s2 & s1;
         y <=  a when s_temp="00" else
                           b when s_temp="01" else
                           c when s_temp="10" else
                           d;
end Behavioral;
```

b) Realizați un multiplexor 4:1 utilizând implementarea prin specificația selectivă WITH / SELECT / WHEN.

Soluție

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux_b is
   Port ( a,b,c,d : in std_logic;
         s1, s2 : in std_logic;
         y : out std_logic);
```

```
end mux_b;

architecture Behavioral of mux_b is
   signal s_temp : std_logic_vector(1 downto 0);
begin
         WITH s_temp SELECT
                  y <=  a when "00",
                                    b when "01",
                                    c when "10",
                                    d when OTHERS;

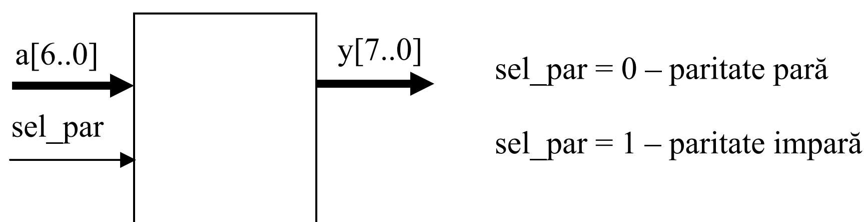
end Behavioral;
```

c) Prezentaţi descrierea hardware a unui corector de paritate în care datele de ieşire vor avea numai paritate pară sau impară. Alegerea parităţii se va face printr-un semnal de intrare **sel_par**.

Soluţie

Corectarea parităţii se realizează prin ataşarea unui bit pe poziţia cea mai semnificativă a informaţiilor de intrare.

În figura de mai jos este prezentată entitatea modulului digital:



De exemplu:

Infirmaţii la intrare **a[6..0]**	Bit paritate **sel_par**	Informaţii la ieşire **y[7..0]**
1001001	0 – paritate pară	**1**1001001
1001001	1 – paritate impară	**0**1001001
0001100	0 – paritate pară	**0**0001100
0001100	1 – paritate impară	**1**0001100

Un număr binar este considerat ca având paritate *pară*, dacă numărul de biţi conţinuţi cu valoarea „1” logic este par (inclusiv valoarea 0).

Un număr binar este considerat ca având paritate *impară*, dacă numărul de biți conținuți cu valoarea „1” logic este impar .

Determinarea parității unui număr binar se realizează prin efectuarea operației logice XOR între toți biții acestuia.

În cazul exemplului nostru, relația de determinare a parității este următoarea:

$$paritate = a[6] XOR\ a[5] XOR\ a[4] XOR\ a[3] XOR\ a[2] XOR\ a[1] XOR\ a[0]$$

Dacă se dorește ca informația de ieșire să aibă paritate pară, atunci la informația de intrare se atașează bitul de paritate corespunzător $y = paritate\ \&\ a;$

Dacă se dorește paritate impară, la informația de intrare se atașează bitul de paritate obținut prin complementare $y = not(paritate)\ \&\ a;$

Programul VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Corect_Parit is
   Port ( a : in std_logic_vector(6 downto 0);
          sel_par : in std_logic;
          y : out std_logic_vector(7 downto 0));
end Corect_Parit;

architecture Behavioral of Corect_Parit is
signal paritate : std_logic;
begin

        paritate <= a(6)XOR a(5)XOR a(4)XOR a(3)XOR a(2)XOR a(1)XOR a(0);

        with sel_par select
                y <= paritate & a when '0',
                        (not paritate) & a when others;

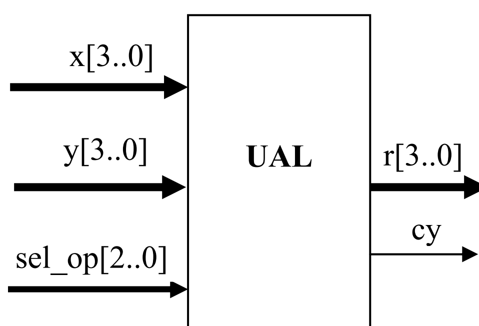
end Behavioral;
```

d) Descrieți în limbajul VHDL o structură digitală ce implementează o unitate aritmetico-logică (UAL) ce procesează doi operanzi de intrare **x**, **y** reprezentați pe 4 biți şi obține rezultatul **r** prezentat tot pe patru biți. Operațiile realizate de către UAL sunt prezentate în tabelul de mai jos:

sel_op[2..0]	**operație**	**funcție**	**unitate**
000	r <= x	Transferă x la ieşire	Aritmetică
001	r <= y	Transferă y la ieşire	
010	r <= x + y	Adună x cu y (fără transport)	
011	cy,r <= x + y	Adună x cu y (cu transport)	
100	r <= x AND y	ŞI logic	Logică
101	r <= x OR y	SAU logic	
110	r <= NOT x	Complement x	
111	r <= NOT y	Complement y	

Soluție

Entitatea unității aritmetico-logice este simbolizată în figura de mai jos:



Selecția între unitatea logică şi unitatea aritmetică se face cu bitul cel mai semnificativ al semnalului **sel_op**. Biții **sel_op[0]** şi **sel_op[1]** sunt utilizați în selecția funcțiilor logice sau aritmetice corespunzătoare celor două unități de calcul.

Soluția prezentată în programul următor este realizată integral în domeniul concurent. Cele două unități de calcul (aritmetic şi logic) operează, de asemenea, în paralel realizând calculele cu cei doi operanzi de intrare **x**, respectiv **y**.

Implementările acestor operații sunt posibile datorită includerii pachetului de funcții **std_logic_unsigned** din biblioteca **IEEE**. În cadrul programului mai sunt utilizate două semnale care fac legătura dintre unitatea aritmetică, respectiv unitatea logică cu multiplexorul de la ieşire.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity UAL is
   Port ( x, y : in std_logic_vector(3 downto 0);
          sel_op : in std_logic_vector(2 downto 0);
          r : out std_logic_vector(3 downto 0);
          cy : out std_logic);
end UAL;

architecture Behavioral of UAL is
        signal t_arith : std_logic_vector(4 downto 0);
        signal t_logic : std_logic_vector(3 downto 0);

begin

        --descrierea unitatii aritmetice
        with sel_op(1 downto 0) select
                t_arith <= '0'& x when "00",
                           '0'& y when "01",
                           '0'& x + '0'& y when "10",
                           '0'& x - '0'& y when others;

        --descrierea unitatii logice
        with sel_op(1 downto 0) select
                t_logic <= x and y when "00",
                           x or y when "01",
                           not x when "10",
                           not y when others;

        --multiplexarea intre unitatea logica si unitatea aritmetica

        r  <= t_arith(3 downto 0)  when sel_op(2)='0' else
                  t_logic;
  cy <= t_arith(4) when sel_op(2)='0' else
                  '0';

end Behavioral;
```

Cele două unități de calcul, aritmetic şi logic, au la bază specificații concurente selective. Pentru multiplexor s-a folosit o specificație concurentă condițională.

În unitatea aritmetică s-a lucrat cu operanzii pe 5 biți astfel încât să se obțină rezultatul operației pe 4 biți şi transport pe 1 bit. Operanzii au fost măriți cu un bit (de la 4 biți la 5 biți) prin operatorul de concatenare "**&**".

De exemplu, semnalul **x** este reprezentat pe 3 biți, dar scriindu-l sub forma **'0' & x** rezultă un semnal pe 4 biți, bitul cel mai semnificativ fiind 0 logic.

Bitul **c**$_y$ este extras numai atunci când are loc operația aritmetică, adică **sel_par[2]='1'**.

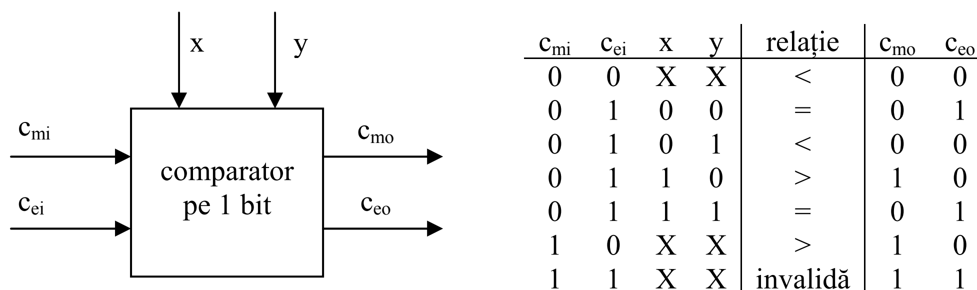
e) Realizați implementarea în cod VHDL a unui modul digital care compară două valori binare **x**, **y** reprezentate pe 4 biți şi semnalizează relația dintre acestea.

Soluție

Realizarea unui comparator direct pe mai mulți biți utilizând ecuații booleene este dificilă. Pentru simplificarea soluției, se foloseşte un comparator general pe un singur bit ce poate fi plasat în cascadă cu alte module de acelaşi tip, formând comparatorul pe mai mulți biți. Comparația se va face secvențial începând cu biții cei mai semnificativi.

Etapa 1. Realizarea comparatorului pe un singur bit.

Comparatorul pe un singur bit trebuie să poată semnaliza următoarele relații dintre operatorii x şi y : $x < y$, $x > y$ şi $x = y$. În plus, comparatorul conține două porturi de intrare, c_{mi} şi c_{ei}, care indică starea modulului comparator anterior (pentru biții cu grad mai mare de semnificație). Dacă $c_{mi} = 1$ atunci $x > y$, iar dacă $c_{ei} = 1$, între operatorii de intrare există egalitate. Acest modul conține şi două porturi de ieşire, c_{mo} şi c_{eo} prin care se semnalizează relațiile de ordine dintre operatorii de intrare către modulele de comparație ulterioare (pentru biții cu grad mai mic de semnificație). Modulul digital are următoarea schemă:



c_{mi}	c_{ei}	x	y	relație	c_{mo}	c_{eo}
0	0	X	X	$<$	0	0
0	1	0	0	$=$	0	1
0	1	0	1	$<$	0	0
0	1	1	0	$>$	1	0
0	1	1	1	$=$	0	1
1	0	X	X	$>$	1	0
1	1	X	X	invalidă	1	1

Ecuațiile booleene simplificate rezultate din tabelul de adevăr sunt următoarele:

$$c_{mo} = c_{mi} + c_{ei} \cdot x \cdot \overline{y}$$

$$c_e = c_{mi} \cdot c_{ei} + c_{ei} \cdot \overline{x} \cdot \overline{y} + c_{ei} \cdot x \cdot y$$

Programul sursă în VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity comp1 is
        port(
                x : in STD_LOGIC;
                y : in STD_LOGIC;
                cmi : in STD_LOGIC;
                cei : in STD_LOGIC;
                cmo : out STD_LOGIC;
                ceo : out STD_LOGIC
                );
end comp1;

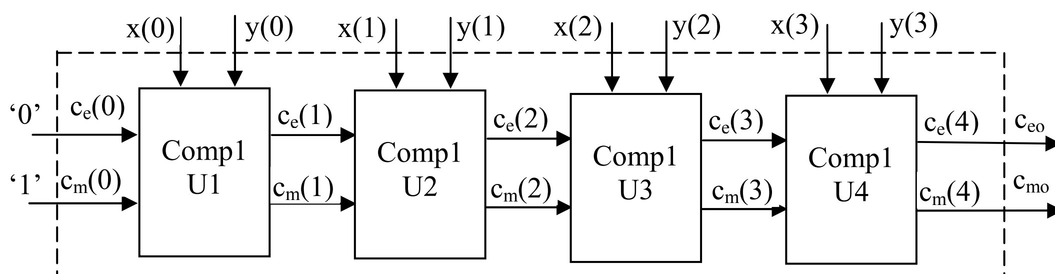
architecture comp1 of comp1 is
begin

        cmo <= cmi or (cei and x and (not y));
       ceo <= (cmi and cei)or(cei and(not x)and (not y))or(cei and x and y);

end comp1;
```

Etapa 2. Realizarea comparatorului pe 4 biți.

O primă soluție constă în realizarea comparatorului pe 4 biți utilizând comparatoare pe un singur bit conectate în cascadă, după cum se prezintă în figura de mai jos:



Cascadarea se realizează prin declararea unei componente **comp1** şi interconectarea (instanțierea) acesteia prin intermediul specificației PORT MAP. Programul sursă pentru descrierea comparatorului pe 4 biți este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
entity comp4 is
        port(
                x : in STD_LOGIC_VECTOR(3 downto 0);
                y : in STD_LOGIC_VECTOR(3 downto 0);
                cmo : out STD_LOGIC;
                ceo : out STD_LOGIC
                );
end comp4;

architecture comp4 of comp4 is
component comp1
port(
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        cmi : in STD_LOGIC;
        cei : in STD_LOGIC;
        cmo : out STD_LOGIC;
        ceo : out STD_LOGIC
        );
end component;
signal cm,ce:std_logic_vector(4 downto 0);
begin
  cm(0)<='0';
  ce(0)<='1';
  U1: comp1 port map(x(0),y(0),cm(0),ce(0),cm(1),ce(1));
  U2: comp1 port map(x(1),y(1),cm(1),ce(1),cm(2),ce(2));
  U3: comp1 port map(x(2),y(2),cm(2),ce(2),cm(3),ce(3));
  U4: comp1 port map(x(3),y(3),cm(3),ce(3),cm(4),ce(4));
  cmo<=cm(4);
  ceo<=ce(4);
end comp4;
```

O a doua soluție, optimizată, ce permite generalizarea constă în realizarea comparatorului pe *n* biți utilizând specificația FOR-GENERATE.

În structura comparatorului pe 4 biți se observă că este utilizat doar modulul **comp1** şi este interconectat repetitiv de patru ori. Acest lucru dă posibilitatea ca descrierea structurală a comparatorului pe 4 biți să fie realizată cu specificația FOR GENERATE ca în programul de mai jos:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity comp4 is
        port(
                x : in STD_LOGIC_VECTOR(3 downto 0);
```

```
                y : in STD_LOGIC_VECTOR(3 downto 0);
                cmo : out STD_LOGIC;
                ceo : out STD_LOGIC
                );
end comp4;

architecture comp4 of comp4 is
component comp1
port(
        x : in STD_LOGIC;
        y : in STD_LOGIC;
        cmi : in STD_LOGIC;
        cei : in STD_LOGIC;
        cmo : out STD_LOGIC;
        ceo : out STD_LOGIC
        );
end component;
signal cm,ce:std_logic_vector(4 downto 0);

begin

cm(0)<='0';
ce(0)<='1';

unit: for i in 0 to 3 generate
U1: comp1 port map(x(i),y(i),cm(i),ce(i),cm(i+1),ce(i+1));
end generate;

cmo<=cm(4);
ceo<=ce(4);

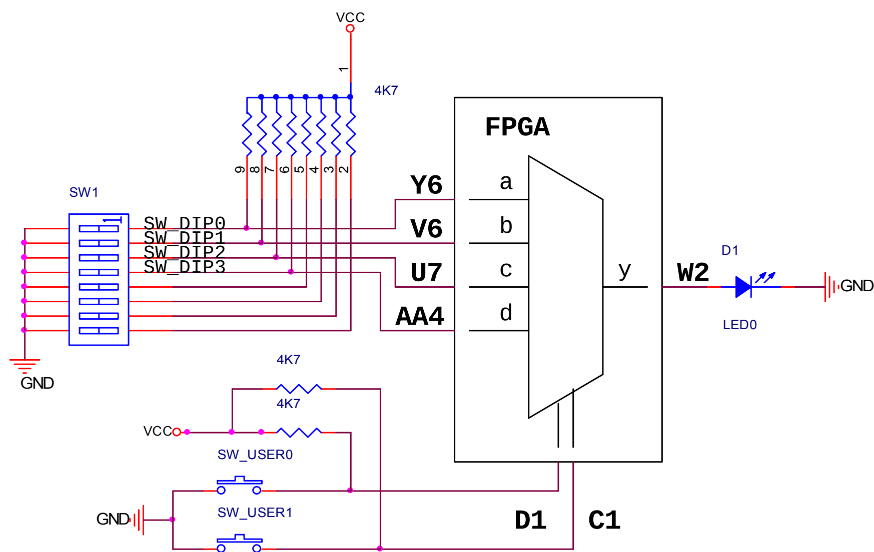
end comp4;
```

3. Exerciții

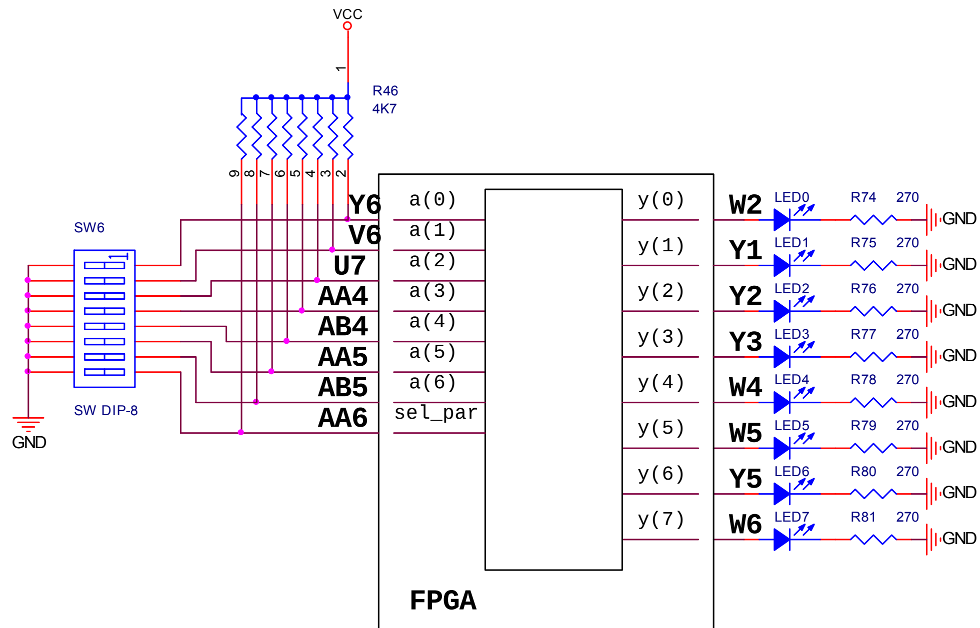
1. Se citesc informațiile prezentate anterior şi se exemplifică pe calculatoarele existente;
2. Se vor implementa programele din exemplele a) … e) folosind schemele hardware de mai jos (2.1 … 2.4). Implementarea se va realiza în următorii paşi:
   - pentru fiecare aplicație în parte se realizează un proiect nou;
   - se introduc sursele VHDL;
   - se simulează logic proiectul;
   - se creează fişierele de constrângeri ale pinilor după schemele hardware de mai jos (2.1 … 2.4);
   - se realizează sinteza acestora;
   - se vizualizează schemele logice, respectiv tehnologice;
   - se simulează modulul rezultat după sinteză;
   - se implementează proiectul;
   - se vizualizează rutarea structurii fizice;
   - se realizează simularea Post-Place & Post Route a modulului;
   - se configurează circuitul fizic.

Notă: Circuitul configurabil FPGA este de tipul **3s400fg456** cu speed grade **-4**

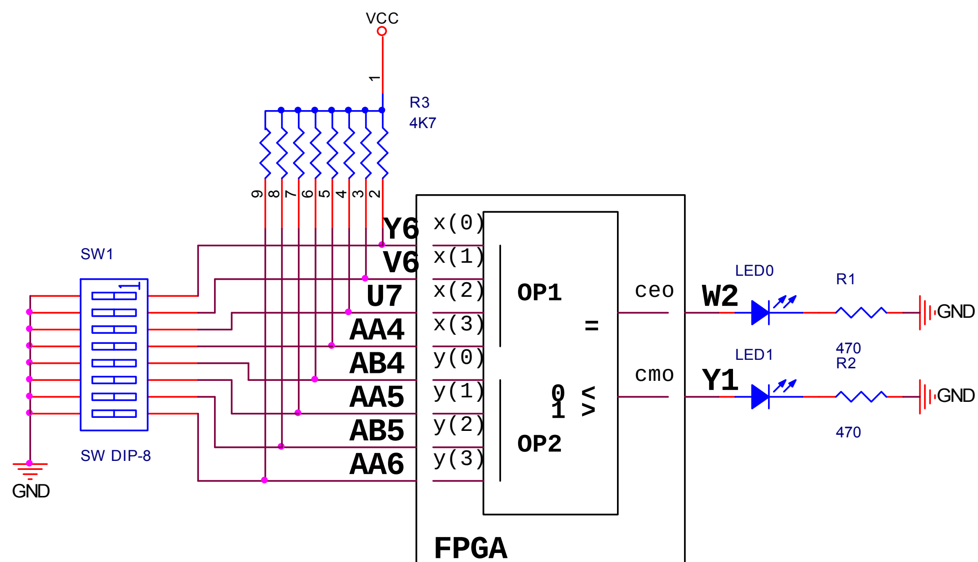
2.1. Multiplexor 4:1 (exemplele a şi b)



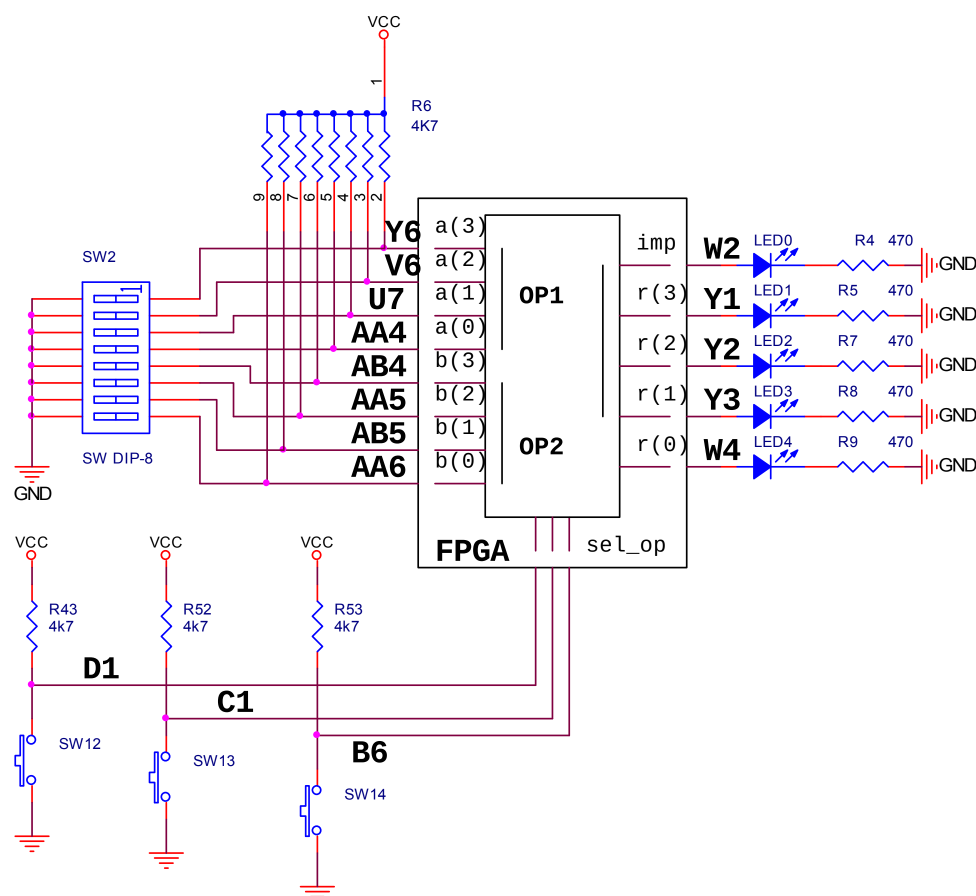
2.2. Corectorul de paritate (exemplul c)



2.3. Comparatorul pe 4 biți (exemplul d)



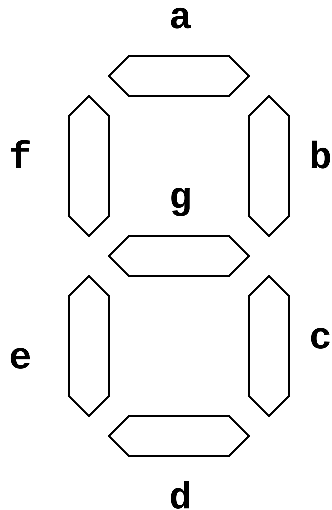
2.4. Unitate aritmetico-logică (exemplul e)

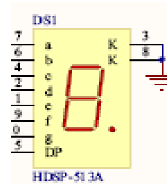


Legăturile dintre pinii fizici ai structurii reconfigurabile şi porturile din entitățile modulelor descrise în VHDL sunt specificate în schemele electrice anterioare.

Componentele din schemele electrice (butoane, LED-uri, rezistențe) sunt prezente în schema sistemului de dezvoltare. Entitatea modulului digital se regăseşte în chenarul notat: **FPGA**.

3. Să se realizeze un decodor pe 4 biți ce va permite afișarea valorii în format zecimal a informației binare de intrare, sub forma numerelor de la 0 la 9, utilizând un digit cu șapte segmente. Valorile binare de intrare sunt aplicate cu ajutorul butoanelor USER0, USER1, USER2, USER3 de pe sistemul de dezvoltare.

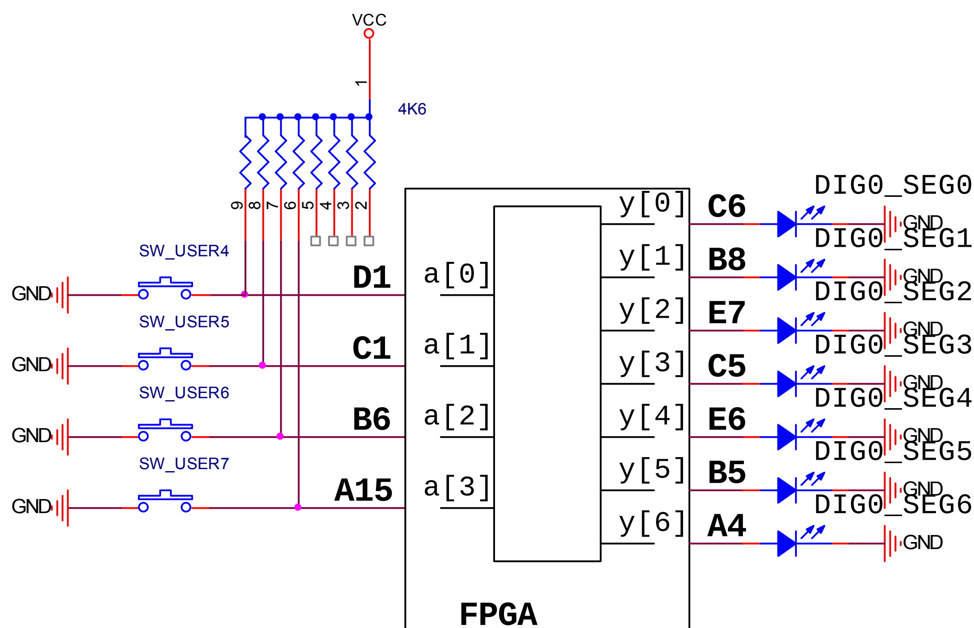




Corespondența dintre liniile circuitului fpga și semnalele ce acționează cele 7 segmente este următoarea:

D_7	D_6	D_5	D_5	D_3	D_2	D_1	D_0
dp	g	f	e	d	c	b	a

dp –dot point



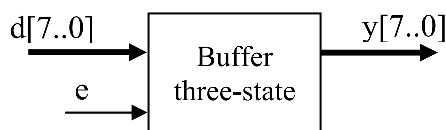
Să se realizeze acest modul prin două metode: descriere de tip flux de date, respectiv descriere structurală.

4. Să se realizeze în limbajul VHDL un buffer inversor pe 8 biți cu ieşirile de tip three-state (tree-state sau TS) utilizând numai specificații concurente.

e	y
0	Z
1	d

Numele de buffer three-state provine de la ieşirea acestuia care poate lua una din cele trei stări posibile: '1' logic, '0' logic sau 'Z', aceasta fiind starea de înaltă impedanță - HiZ. În general buffer-ele de tip three-state sunt utilizate pentru conectarea unor module digitale la magistralele de date. Un etaj de ieşire three-state permite cuplarea în paralel a mai multor ieşiri. Într-o astfel de configurație, fiecare ieşire poate fi activă la un moment dat (are acces la magistrala de ieşire comună), dar celelalte ieşiri trebuie să fie în starea de înaltă impedanță.

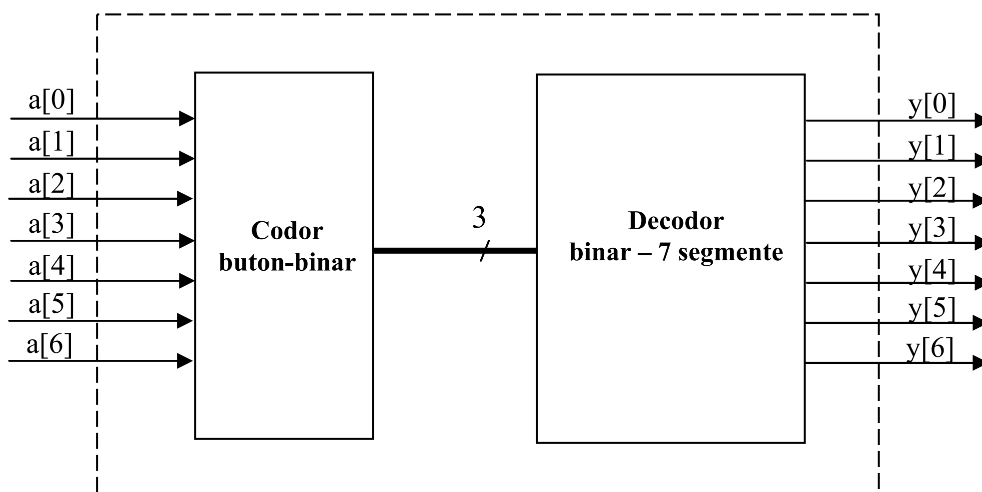
Tabelul de adevăr şi porturile de interconectare ale unui astfel de buffer sunt date în figura de mai jos:



5. Să se realizeze în limbaj VHDL un modul digital care afişează numărul de biți aflați în starea 1 logic dintr-o informație pe 7 biți ce se aplică la un port de intrare.

Corelarea cu schema fizică a porturilor de intrare/ieşire se va face după schema electrică a sistemului reconfigurabil.

Intern, modulul digital va fi realizat din două module descrise cu specificații concurente şi interconectate prin descriere structurală.



Capitolul 3
Programarea structurilor hardware reprogramabile prin descrieri cu specificații secvențiale

În acest capitol sunt prezentate implementări cu descrieri secvențiale ale modulelor digitale ce conțin logică sincronă sau asincronă. Sunt realizate mai multe aplicații în care sunt cuprinse o parte din specificațiile secvențiale ce vor fi sintetizate şi implementate pe sistemul reconfigurabil din cadrul laboratorului. În final sunt propuse probleme din aceeaşi categorie în vederea implementării.

1. Breviar teoretic

1.1. Semnalele şi variabilele în domeniul secvențial

Transportul datelor în VHDL poate fi realizat prin semnale sau prin variabile. În timp ce semnalele pot fi declarate în domeniul concurent, variabilele pot fi declarate numai în domeniul secvențial. Semnalul poate fi utilizat global, în domeniile concurente şi secvențiale, iar variabila este numai locală domeniului secvențial. Domeniul secvențial poate fi descris prin procese, funcții, respectiv proceduri.

Un proces defineşte o listă de specificații secvențiale prin care se descrie comportamental o anumită structură dintr-un modul digital. Într-un proces, dacă nu este necesară utilizarea unui semnal, se poate utiliza o variabilă. Semnalul nu poate fi declarat într-un proces. Valoarea unui semnal este afectată numai la ieşirea din proces de ultima atribuire, chiar dacă asupra lui s-au făcut alte atribuiri în timpul procesului.

Declarația unei variabile se poate face prin utilizarea următoarei sintaxe:

```
variable nume_variabilă : tip;
variable nume_variabilă : tip := valoare_inițială;
```

Atribuirea unei variabile se realizează după sintaxa:

```
variable_name := expression ;
```

1.2. Introducerea unui proces

Specificațiile secvențiale pot fi introduse prin intermediul clauzei „PROCESS". Procesele sunt activate printr-o listă de senzitivități. Dacă lista de senzitivități lipsește, activarea procesului se realizează prin specificația WAIT.

De menționat faptul că procesele conțin descrieri secvențiale, dar între ele sunt concurente.

Sintaxă:

```
[eticheta:] PROCESS (lista de senzitivitati)
        [VARIABILE nume: tip [dimensiune] [:= valoare_initiala;]]
BEGIN
        (cod secvential)
END PROCESS [eticheta];
```

Între clauzele **process** şi **begin** se găseşte zona declarativă în care pot fi declarate variabile, tipuri, subprograme, atribute, etc. Zona de descriere secvențială este definită între clauzele **begin** şi **end**.

1. 3. Specificații secvențiale

Specificația IF

Specificația IF este utilizată în structuri condiționale şi are următoarea sintaxă:

```
IF conditie THEN specificatii_secventiale;
ELSIF conditie THEN specificatii_secventiale;
…..
ELSE specificatii_secventiale;
END IF;
```

Datorită influenței puternice a mediilor de programare software (de exemplu C++, PASCAL, etc.), tendința programatorilor este de a utiliza structurile condiționale în descrierea comportamentului unui modul digital, fără a mai face recurs la descrierile de tip flux de date prin ecuații booleene sau alte specificații care ocupă o arie hardware mult mai mică. Totuşi, utilizarea specificației IF nu afectează, în principiu, structura hardware foarte mult. Explicația este dată de faptul că în procesul de sinteză se produce o optimizare a ecuațiilor logice şi evitându-se astfel mărirea complexității hardware nejustificate. Totuşi, este indicat ca imbricarea specificațiilor **IF** să nu se facă pe prea multe nivele.

IF este o specificație secvențială care nu poate fi utilizată în zona concurentă a unei arhitecturi şi este diferită de specificația **IF GENERATE** din domeniul concurent.

Exemplu de utilizare a unei structuri condiționale IF

```
IF (reset = '1') THEN data_out <= (others => ,1')
ELSIF (clk='1' AND clk'event) THEN data_out <= data_in;
ELSE data_out <= (others => ,Z');
END IF;
```

În exemplul de mai sus dacă semnalul **RESET** este activ în 1 logic, semnalul de ieşire va pune toate liniile acestuia în 1 logic. Dacă semnalul **RESET** nu este activ şi a avut loc un eveniment pe frontul pozitiv al semnalului **CLK**, semnalul de ieşire primeşte valorile semnalului de intrare. În caz contrar semnalul de ieşire este trecut în starea de înaltă impedanță.

Specificația WAIT

Specificația **WAIT** este utilizată în cazurile în care procesul nu are o listă de senzitivități. Specificația poate fi utilizată sub trei forme, după cum se prezintă în sintaxele următoare:

```
WAIT UNTIL conditie_semnal
WAIT ON semnal1 [, semnal2, …];
WAIT FOR time;
```

Prima sintaxă este utilizată în general pentru modelele digitale sincrone, fiind mai puțin folosită la cele asincrone. Acest lucru se datorează faptului că prin specificația WAIT UNTIL este introdusă o condiție asupra unui semnal de care nu se poate trece până când condiția respectivă nu este îndeplinită.

Cea de-a doua sintaxă este utilizată atunci când sunt monitorizate mai multe semnale. Procesul devine activ numai când unul din semnalele aflate în lista specificației WAIT ON îşi schimbă starea.

În final, ultima specificație WAIT FOR este introdusă numai pentru simularea modulelor digitale în fişierele test. Această specificație nu este sintetizabilă.

De exemplu: WAIT FOR 100ns

Specificațiile WAIT sunt plasate imediat după BEGIN în cadrul unui proces.

Specificația CASE

Specificația CASE este utilizată pentru selectarea unei alternative în funcție de valoarea unei expresii.

```
CASE identificator IS
    WHEN value => atribuire;
    WHEN value => atribuire;
    …
```

```
        WHEN OTHERS => atribuire;
    END CASE;
```

Specificația CASE evaluează o expresie şi selectează una din alternative în concordanță cu valoarea acesteia. Expresia de evaluare poate fi de natură discretă (numere întregi) sau şir de caractere. CASE conține o listă de alternative care încep cu specificația WHEN. Este urmată de valoarea corespunzătoare alternativei respective şi de specificațiile secvențiale care trebuie executate în cazul în care este aceasta este aleasă.

Clauza OTHERS este folosită atunci când sunt luate în considerare şi alte valori ale identificatorului ce nu sunt prevăzute în alternativele cuprinse în WHEN.

Specificația LOOP

Specificația LOOP este utilizată pentru repetarea unor secvențe de cod VHDL, după o anumită condiție WHILE/LOOP sau repetitiv cu specificația FOR/LOOP.

FOR/LOOP – bucla este repetată de un număr de ori predefinit care nu se mai poate schimba după intrarea în aceasta.

```
[eticheta:] FOR identificator IN interval LOOP
        (specificatii secventiale)
END LOOP [eticheta:];
```

WHILE-LOOP – bucla este repetată până când nu mai este îndeplinită condiția.

```
[eticheta:] WHILE conditie LOOP
        (specificatii secventiale)
END LOOP [eticheta:];
```

EXIT – este utilizată pentru terminarea forțată a unei bucle.

```
[eticheta:] EXIT [eticheta] [WHEN conditie];
```

NEXT – este folosită pentru omiterea unui pas într-o buclă.

```
[eticheta:] NEXT [eticheta bucla] [WHEN conditie];
```

2. Aplicații

a) Folosind limbajul VHDL, să se realizeze un numărător BCD către înainte având porturile de intrare/ieşire precizate în figura de mai jos:



Soluție

Modulul digital conține doar un semnal de intrare **clk** şi la ieşire un semnal pe patru biți denumit **data_out**. Exemplul realizat utilizează specificația **IF THEN**.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity numarator is
	port(
		clk : in STD_LOGIC;
		data_out : out STD_LOGIC_VECTOR(3 downto 0)
		);
end numarator;

architecture numarator of numarator is
begin
	PROCESS (clk)
		VARIABLE  temp_num:  STD_LOGIC_VECTOR(3  downto
0):="0000";
	BEGIN
		IF (clk'event AND clk='1')THEN
			temp_num:=temp_num+1;
			IF (temp_num="1010") THEN
				temp_num:=(OTHERS => '0');
			END IF;
		END IF;

		data_out<=temp_num;
	END PROCESS;
end numarator;
```

Descrierea numărătorului este de tip comportamental prin specificații secvențiale în cadrul unui proces. Procesul este activat numai la apariția unui eveniment pe frontul pozitiv al semnalului **clk**, fiind plasat în lista de senzitivități a acestuia. În zona declarativă a procesului este introdusă variabila **temp_num**, cu valoarea inițială 0 şi ce contorizează stările la numărare. Numărarea se efectuează pe frontul crescător a semnalului de ceas prin linia de cod:

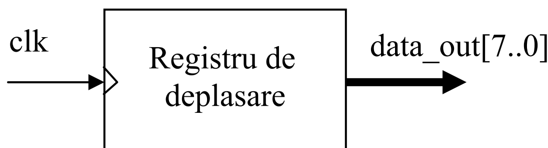
```
IF (clk'event AND clk='1')THEN
```

Menținerea în domeniul zecimal a valorii variabilei **temp_num** (în gama de la 0 la 9) se obține prin linia de cod:

```
IF (temp_num="1010") THEN
```

Cum semnalele se pot actualiza numai la ieşirea din proces, atribuirea semnalului de ieşire **data_out** cu variabila **temp_num** s-a făcut la sfârşitul procesului.

b) Folosind limbajul VHDL, să se realizeze un registru de deplasare în inel pe 8 biți, de la dreapta la stânga, acționat pe frontul pozitiv al semnalului de ceas **clk**. Valoarea inițială a registrului este „00000001". La fiecare impuls de ceas deplasarea se face cu un bit, iar bitul cel mai puțin semnificativ rămâne în starea 0 logic.



Soluție

Programul VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity process_shift is
        port(
                clk : in STD_LOGIC;
                data_out : out STD_LOGIC_VECTOR(7 downto 0)
                );
end process_shift;

architecture process_shift of process_shift is
begin
```

```
PROCESS
        VARIABLE temporar:STD_LOGIC_VECTOR(7 downto
0):="00000001";
        BEGIN
                WAIT UNTIL (clk'event AND clk='1');
                        data_out<=temporar;
                        temporar:=temporar(6 downto 0)&'0';
        END PROCESS;
end process_shift;
```

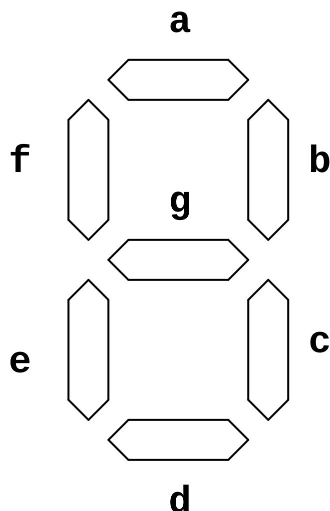
Procesul nu prezintă o listă de senzitivități datorită utilizării specificației WAIT UNTIL. Respectiva specificație nu permite intrarea în proces dacă nu a fost îndeplinită condiția de activare (clk'event AND clk='1'). Variabila **temporar** este deplasată la stânga cu un bit de fiecare dată când este activat procesul. La ieşirea din proces, semnalul **data_out** primeşte valoarea curentă a variabilei **temporar**.

În zona declarativă a procesului, variabila **temporar** este inițializată cu valoarea „00000001".

c) Folosind limbajul VHDL să se realizeze un numărător zecimal cu afişare pe un digit de tip display cu 7 segmente. Numărarea se va face pe frontul pozitiv al semnalului de ceas aplicat, de la 0 la 9.

Soluție

Un display cu 7 segmente este format din leduri ordonate ca în figura de mai jos. Tabela de adevăr pentru translația din codul binar în cod corespunzător celor 7 segmente este următoarea:



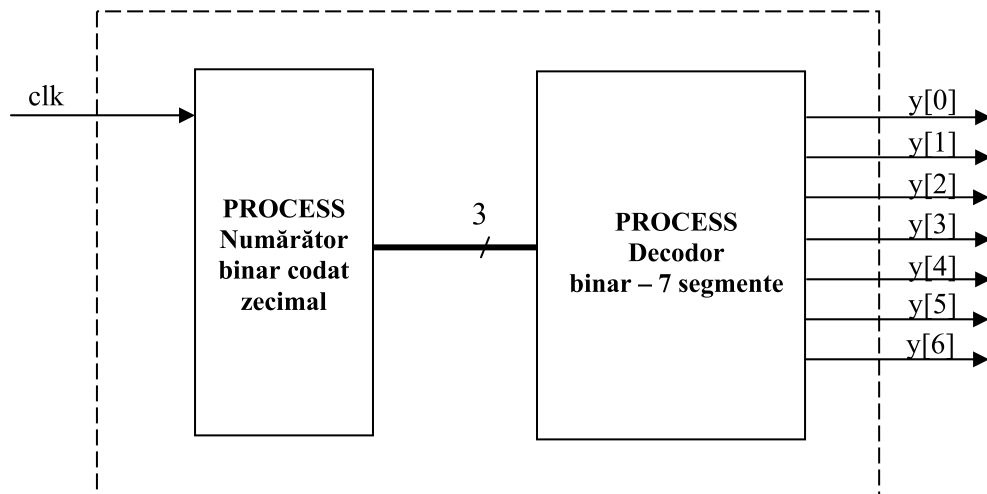
nr. binar	cod 7 segmente
	gfedcba
0000	0111111
0001	0000110
0010	1011011
0011	1001111
0100	1100100
0101	1101101
0110	1111101
0111	0000111
1000	1111111
1001	1101111

Corespondența dintre liniile circuitului fpga şi semnalele ce acționează cele 7 segmente este următoarea:

D_7	D_6	D_5	D_5	D_3	D_2	D_1	D_0
-	g	f	e	d	c	b	a

Ledurile se aprind atunci când pe liniile de comandă ale display-ului pe 7 segmente se generează valoarea “1” logic.

Soluția prezentată exemplifică utilizarea specificației CASE. Entitatea corespunzătoare structurii descrise are următoarea configurație.



Descrierea comportamentală a numărătorului se realizează prin două procese. În primul proces, numărătorul zecimal este descris printr-o logică secvențială. Al doilea proces descrie decodorul binar - 7 segmente, în mod secvențial, printr-o logică combinațională care respectă tabela de adevăr prezentată anterior.

Programul VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity num_7seg is
        port(
                clk : in STD_LOGIC;
                y : out STD_LOGIC_VECTOR(6 downto 0)
                );
end num_7seg;

architecture num_7seg of num_7seg is
        SIGNAL semnal_temp:STD_LOGIC_VECTOR(3 downto 0);

begin
    PROCESS (clk)
     VARIABLE temp_num: STD_LOGIC_VECTOR(3 downto 0):="0000";
    BEGIN
```

```
            IF (clk'event AND clk='1')THEN
                    temp_num:=temp_num+1;
                    IF (temp_num="1010") THEN
                            temp_num:=(OTHERS => '0');
                    END IF;
            END IF;
            semnal_temp<=temp_num;
    END PROCESS;

     PROCESS (semnal_temp)
          VARIABLE iesire_temp: STD_LOGIC_VECTOR(6 downto 0);
     BEGIN
            CASE semnal_temp IS
                    WHEN "0000"=> iesire_temp:="0111111";
                    WHEN "0001"=> iesire_temp:="0000110";
                    WHEN "0010"=> iesire_temp:="1011011";
                    WHEN "0011"=> iesire_temp:="1001111";
                    WHEN "0100"=> iesire_temp:="1100100";
                    WHEN "0101"=> iesire_temp:="1101101";
                    WHEN "0110"=> iesire_temp:="1111101";
                    WHEN "0111"=> iesire_temp:="0000111";
                    WHEN "1000"=> iesire_temp:="1111111";
                    WHEN "1001"=> iesire_temp:="1101111";
                    WHEN OTHERS=> iesire_temp:="ZZZZZZZ";
            END CASE;
            y<=iesire_temp;
     END PROCESS;
end num_7seg;
```

Cele două procese pot fi văzute ca două module digitale separate. Acestea sunt legate între ele printr-un semnal denumit **semnal_temp**.

Primul proces este senzitiv la semnalul de ceas şi incrementează variabila **temp_num** de fiecare dată când apare un front crescător pe **clk**. Dacă variabila **temp_num** a ajuns la valoarea binară 1010 este automat resetă în 0000. La ieşirea din proces, semnalul **semnal_temp** preia valoarea variabilei **temp_num**.

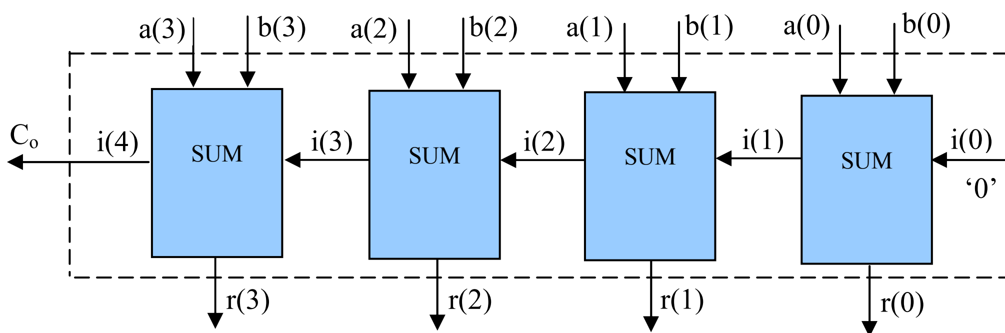
Al doilea proces este asincron şi este senzitiv la semnalul intern **semnal_temp**. În acest proces, prin specificația CASE se face transformarea din codul BCD în 7 segmente. Valoarea corespunzătoare cifrei ce trebuie afişată pe display-ul cu 7 segmente este salvată în variabila **ieşire_temp**. La ieşirea din proces, semnalul **y** preia valoarea variabilei **ieşire_temp**.

d) Folosind limbajul VHDL să se implementeze un sumator binar pe 4 biți, utilizând specificația secvențială LOOP.

Soluție

În limbajul VHDL, un sumator se poate realiza prin mai multe metode. Una dintre metode (descriere stucturală) a fost utilizată la modulul comparator pentru două numere binare, reprezentate pe patru biți, din lucrarea de laborator 2. În acel exemplu s-au utilizat specificații concurente. S-a realizat operația de comparare pentru un singur bit, după care a fost multiplicată de patru ori prin instanțierea de componente.

Soluția ce urmează prezintă o altă metodă de rezolvare (descriere comportamentală) ce utilizează specificații secvențiale. În acest caz nu mai este necesară multiplicarea sumatorului pe un singur bit (vezi cazul comparatorului), ci utilizarea repetitivă, ce are în vedere faptul că adunarea se realizează secvențial, rezultatul obținându-se bit după bit, cu folosirea transportului. Procedura de adunare utilizată este cunoscută sub denumirea Ripple Carry. Schema bloc a modului de implementare a sumatorului digital este prezentată în figura de mai jos:



Ecuațiile booleene ale celulei de sumare pe un singur bit sunt următoarele:

$$r = a \oplus b \oplus c_i$$
$$c_o = a \cdot b + c_i(a \oplus b)$$

Se constată faptul că ieşirea $\mathbf{c_o}$ a celulei k este conectată la intrarea $\mathbf{c_i}$ de la celula k+1 prin semnalul intern **i(k+1)**.

Aplicând ecuațiile anterioare pentru fiecare celulă a sumatorului se obțin relațiile recurente:

$$r(k) = a(k) \oplus b(k) \oplus i(k)$$
$$c_o = i(k+1) = a(k) \cdot b(k) + i(k)(a(k) \oplus b(k))$$

pentru $k \in [0,3]$

Implementarea relațiilor booleene de mai sus poate fi realizată cu specificația LOOP. Sumatorul pe 4 biți este descris prin următorul cod sursă VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity sumator_loop is
	port(
		a : in STD_LOGIC_VECTOR(3 downto 0);
		b : in STD_LOGIC_VECTOR(3 downto 0);
		co : out STD_LOGIC;
		r : out STD_LOGIC_VECTOR(3 downto 0)
		);
end sumator_loop;

architecture sumator_loop of sumator_loop is
begin
	process(a,b)
		variable i:		STD_LOGIC_VECTOR(4 downto 0);
		variable rez:	STD_LOGIC_VECTOR(3 downto 0);
	begin

		i(0):='0';
		for k in 0 to 3 loop
			rez(k):=a(k) xor b(k) xor i(k);
			i(k+1):=(a(k) and b(k)) or (i(k) and (a(k) xor b(k)));
		end loop;
		r<= rez;
		co <= i(4);

	end process;

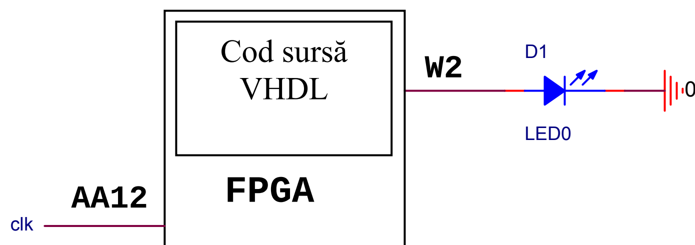
end sumator_loop;
```

Programul de mai sus nu utilizează specificații concurente pentru realizarea sumatorului pe patru biți. Practic, au fost utilizate două relații booleene (pentru rezultat şi pentru transportul de ieşire), care au fost repetate pentru fiecare celulă de sumare elementară, plecând de la bitul cel mai puțin semnificativ.

e) Să se prezinte codul VHDL ce descrie comportamental un temporizator digital folosit pentru aprinderea alternativa a unei diode led, cu frecvența de 1Hz.

Soluție

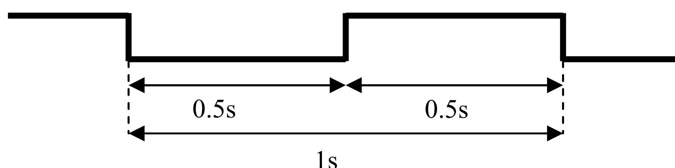
Ideea de realizare a temporizatorului se bazează pe utilizarea unui numărător digital. Semnalul de ceas al numărătorului este preluat de la un oscilator cu cuarț pe frecvența de 50MHz, existent pe sistemul de dezvoltare. Prin divizări repetate se poate obține un semnal cu o frecvență stabilă de 1Hz. Schema bloc de interconectare este dată în figura următoare.



Numărătorul operează cu o constantă dată de raportul dintre frecvența semnalului de intrare **clk** şi o frecvență de două ori mai mare a semnalului de ieşire, conform relației de mai jos:

$$C_{div} = \frac{f_{clk}}{f_{out}} = \frac{50MHz}{2Hz} = 25 \cdot 10^6{}_{(10)} = 17D7840_{(16)}$$

Semnalul de ieşire are frecvența de 1Hz şi factor de umplere de 1/2, astfel încât 0.5s este „1” logic, iar 0.5s „0” logic. Perioada de 0.5s corespunde unui semnal cu frecvența de 2Hz.



În acest caz, constanta de divizare este:
50MHz/2Hz = $25000000_{(10)}$ = $17D7840_{(h)}$, număr reprezentat pe 25 de biți.

Programul sursă VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

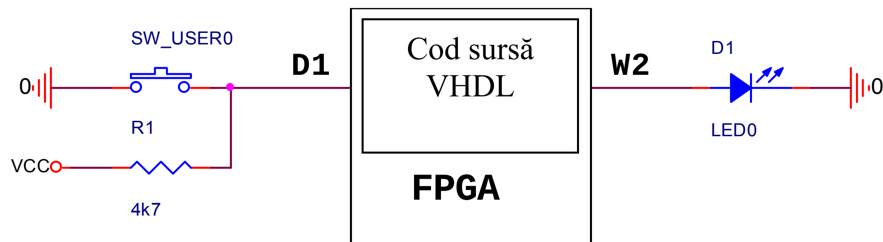
entity temporizator is
        port(
                clk : in STD_LOGIC;
                out_led : buffer STD_LOGIC
                );
```

```
end temporizator;

architecture temporizator of temporizator is
begin
        process (clk,out_led)
                variable Qint: STD_LOGIC_VECTOR (24 downto 0);

                variable temp: STD_LOGIC;
        begin
                temp := out_led;
                if (CLK'event and CLK='1') then
                        Qint:=Qint+1;
                        if (Qint=x"17D7840") THEN
                                Qint:=(others=>'0');
                                out_led <= not temp;
                        end if;
                end if;
        end process;
end temporizator;
```

f) Se consideră un circuit format dintr-un led, o tastă si o structură FPGA conectate ca în schema de mai jos:



Butonul **sw_user0** este conectat la linia **D1** şi ledul **led0** la linia **W2** a structurii de tip FPGA. Se cere să se implementeze un modul digital în cod VHDL care la fiecare apăsare a butonului să schimbe starea ledului din aprins în stins sau invers. <u>Se va ține seama de efectele parazite introduse de contactul butonului</u>.

Soluție

În mod normal, apăsarea şi relaxarea unei taste provoacă generarea unui număr nedeterminabil de impulsuri parazite care pot conduce la interpretări eronate legate de starea acesteia. În schimb, se poate aprecia timpul cât durează acest proces tranzitoriu, şi anume aproximativ 10 ms, atât pentru situația apăsării tastei cât şi la relaxarea acesteia. În limba engleză, termenul folosit pentru acest timp este *debounce time*. Evitarea efectelor parazite menționate se realizează prin folosirea de temporizări adecvate. Procedura de lucru constă în determinarea unui prim impuls apărut ca urmare a contactului făcut la tastă, aşteptarea a 10 ms şi

reluarea testării tastei apăsate, după scurgerea acestui interval de timp. În cazul în care se detectează menținerea contactului la tastă, decizia care se ia este "tastă apăsată" şi se tratează ca atare. În caz contrar, se consideră că inițial a apărut un impuls parazit la apăsarea tastei (sau aceasta nu a fost apăsată ferm), iar decizia care se ia este "tastă neapăsată". Procedura este similară şi pentru tratarea situației de relaxare a respectivei taste cu diferența dată de faptul că după trecerea intervalului de timp de aşteptare menționat, se verifică starea deschisă a contactului.

În condițiile problemei propuse vom putea detecta apăsarea tastei aflate pe linia **D1** prin faptul că la apăsare semnalul de intrare devine "0" logic. Pauza de 10 ms o vom realiza prin introducerea unui modul temporizator comandat de un semnal de ceas **clk** aflat pe pinul AA12 al circuitului SPARTAN3.

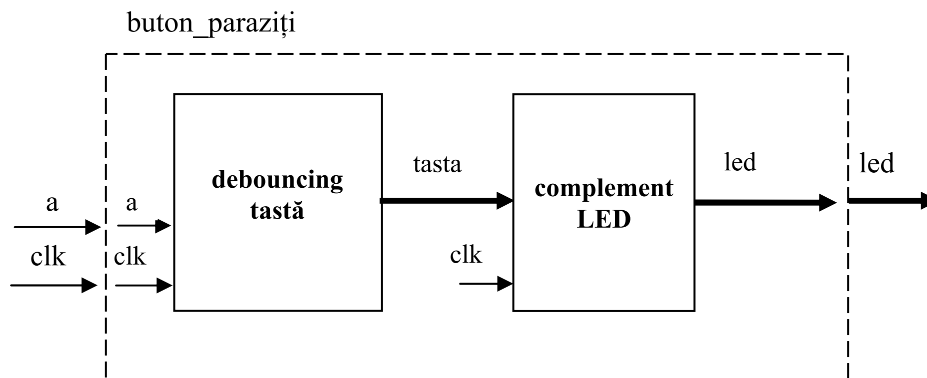
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity buton_paraziti is
	port(
		a : in STD_LOGIC;
		led : out STD_LOGIC;
		clk: in STD_LOGIC
		);
end buton_paraziti;

architecture buton_paraziti of buton_paraziti is
	signal tasta: STD_LOGIC:='1';
begin
	process (clk)
		variable Qint: STD_LOGIC_VECTOR (19 downto
0):=(others =>'0');
		variable val_veche: STD_LOGIC:='0';
	begin
		if (CLK'event and CLK='1') then
			if (a xor val_veche)='1' then
				Qint:=(others=>'0');
				val_veche:=a;
			else
				Qint:=Qint+'1';
				if ((Qint="00111111111111111111")
and ((val_veche xor a)='0')) THEN
					tasta <= val_veche;
				end if;
			end if;
		end if;
	end process;

	process (clk)
```

```
                variable temp: std_logic:='0';
                variable tasta_veche: std_logic:='1';
        begin
                if (CLK'event and CLK='1') then
                        if ( tasta='0' and tasta_veche='1') then
                                temp:=not temp;
                        end if;
                        tasta_veche :=tasta;
                end if;
                        led <=temp;
        end process;
end buton_paraziti;
```

Programul este format din două procese. În primul proces este implementată testarea butonului prin tehnica descrisă anterior. Este utilizat un numărător care se resetează la fiecare apăsare sau relaxare a tastei. Dacă o tastă este apăsată sau relaxată, noua stare a acesteia diferă de cea veche şi atunci se resetează contorul numărătorului. Dacă tasta ținută într-o stare mai mult de 10ms, terminarea temporizării conduce la atribuirea valorii logice a tastei la semnalul **tasta**.



Practic, semnalul **tastă** preia valoarea logică a tastei, dacă aceasta a fost menținută minimum 10ms în aceeaşi stare.

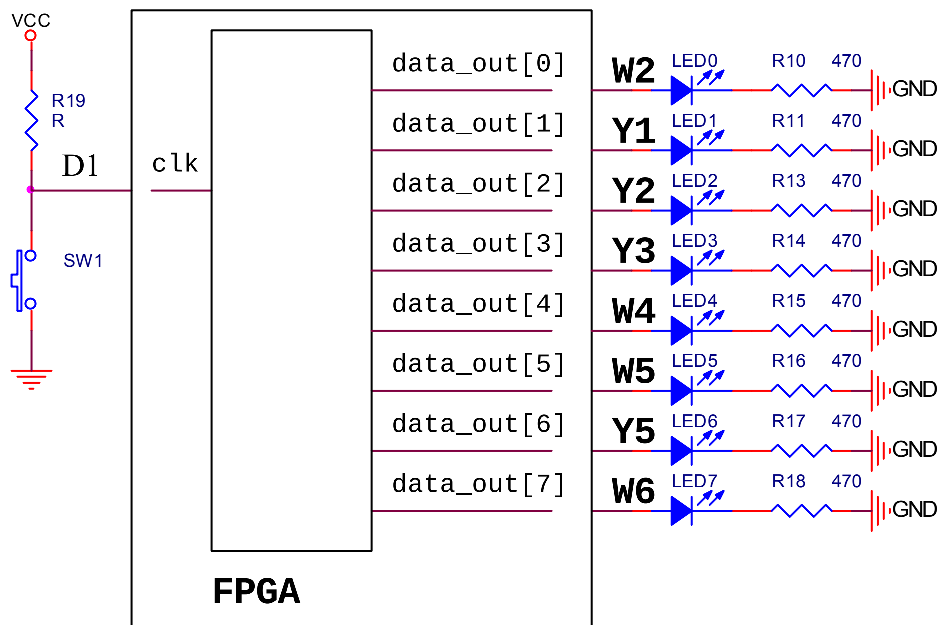
În al doilea proces se realizează complementarea semnalului de comandă al ledului. La tranziția pe frontul crescător al semnalului **tastă**, semnalul de ieşire este complementat prin intermediul variabilei **temp**. S-a folosit variabila **temp** care atribuie valoarea semnalului **led,** după care o returnează acestuia, dar complementată. Pentru evitarea creării unui semnal de ieşire de tip *buffer*, s-a declarat aceasta variabila asupra căruia se fac operațiile de complementare, ce se atribuie ulterior ieşirii **led**.

3. Exerciții

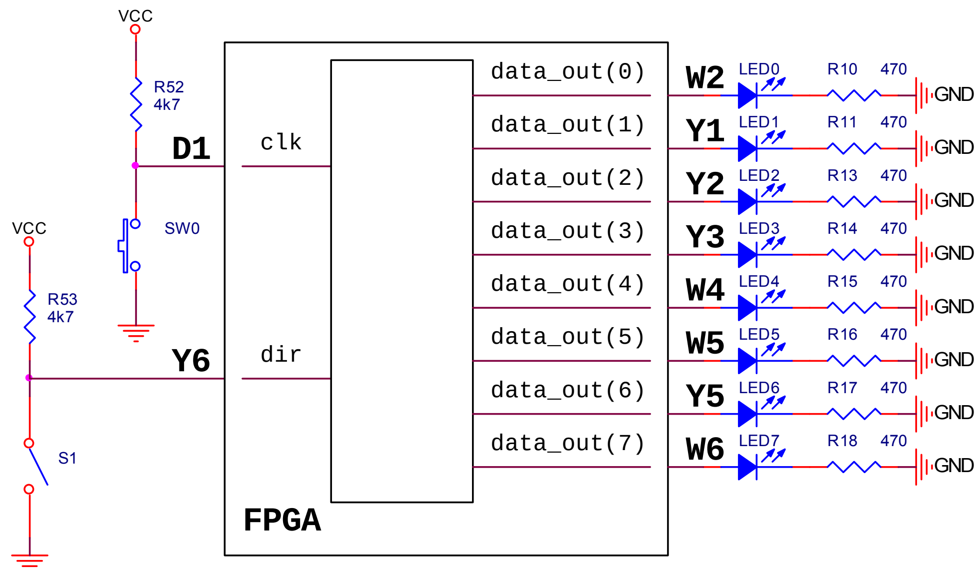
1. Se citesc informațiile prezentate anterior şi se exemplifică pe calculatoarele existente;
2. Se vor implementa structurile propuse prin exemplele anterioare a) … f). Implementarea se va realiza în următorii paşi:
   - pentru fiecare aplicație în parte se realizează un proiect nou;
   - se introduc sursele VHDL;
   - se simulează logic proiectul;
   - se creează fişierele de constrângeri ale pinilor după schemele hardware corespunzătoare;
   - se realizează sinteza acestora;
   - se simulează modulul rezultat după sinteză;
   - se implementează proiectul;
   - se realizează simularea Post-Place & Post Route a modulului;
   - se configurează circuitul fizic.

Notă: Circuitul configurabil FPGA este de tipul **3s400fg456** cu speed grade **-4**
Folosindu-vă de exemplele prezentate anterior sa se rezolve următoarele probleme:

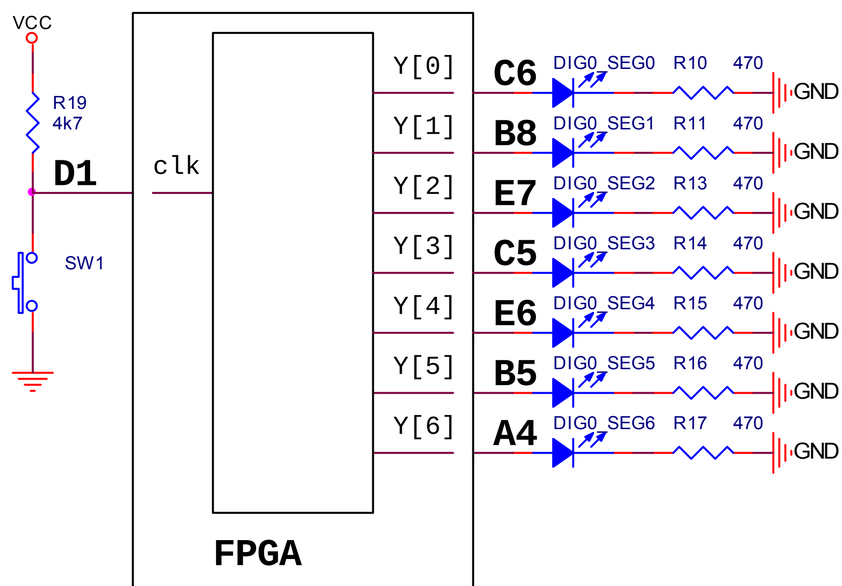
2. Să se implementeze un registru cu deplasare serială stânga (către biții cu grad mai mare de semnificație), în inel, având 8 ieşiri paralele ce acționează leduri, conform schemei de mai jos. Starea inițială a registrului este „00000001”, corespunzătoare ledului de pe ieşirea 0 aprins, celelalte fiind stinse. Schimbarea stării registrului se face la apăsarea tastei SW1.



3. Să se modifice modulul anterior prin adăugarea unui semnal de intrare care să schimbe direcția de deplasare în inel, denumit **dir**. Dacă **dir=1**, deplasarea se va realiza la stânga (către biții cu grad mai mare de semnificație), iar pentru **dir=0**, deplasarea se va realiza la dreapta (către biții cu grad mai mic de semnificație). Schema electrică este următoarea:

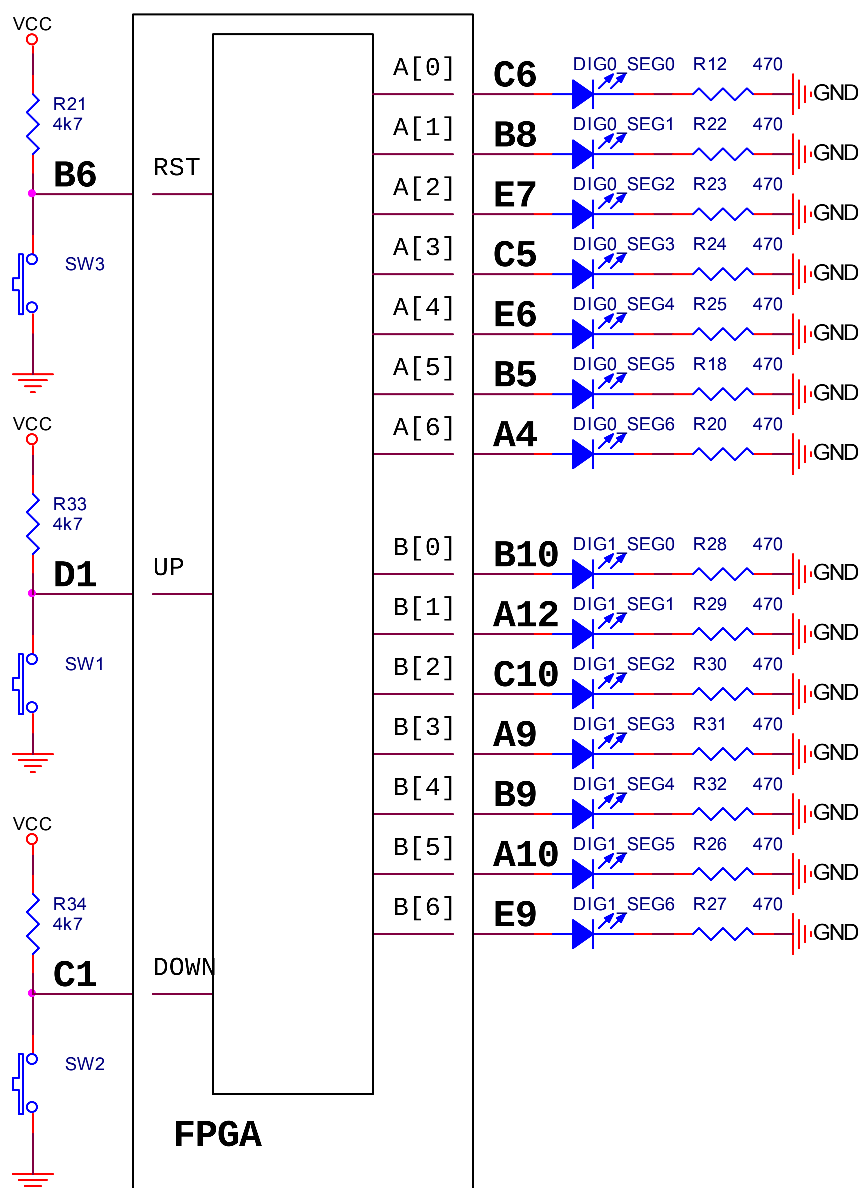


4. Să se implementeze un numărător zecimal cu afişarea stării pe un digit având 7 segmente led în sistemul de dezvoltare după schema următoare. Numărătorul va fi acționat de tasta SW1. Soluția cerută face abstracție de fenomenul de debouncing.

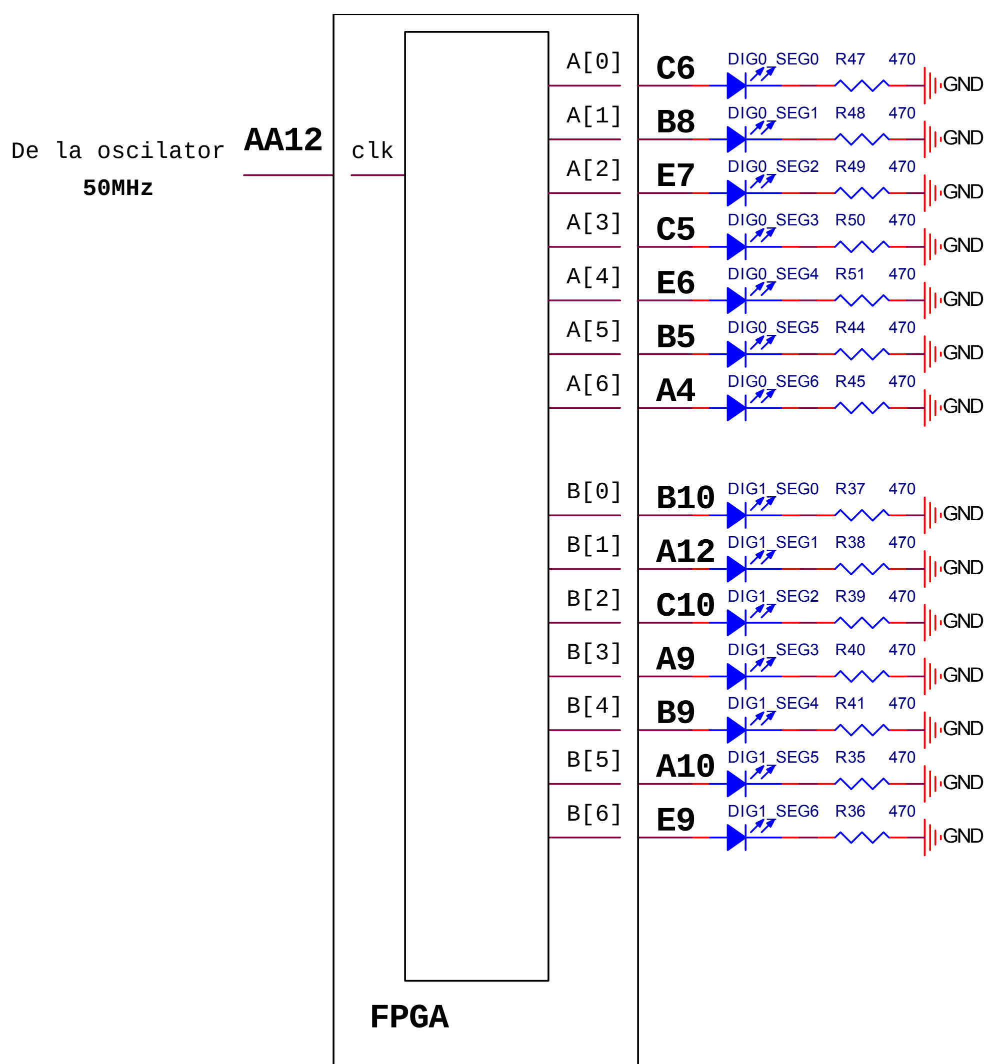


5. Să se rezolve problema anterioară ținând seama de efectul de debouncing.

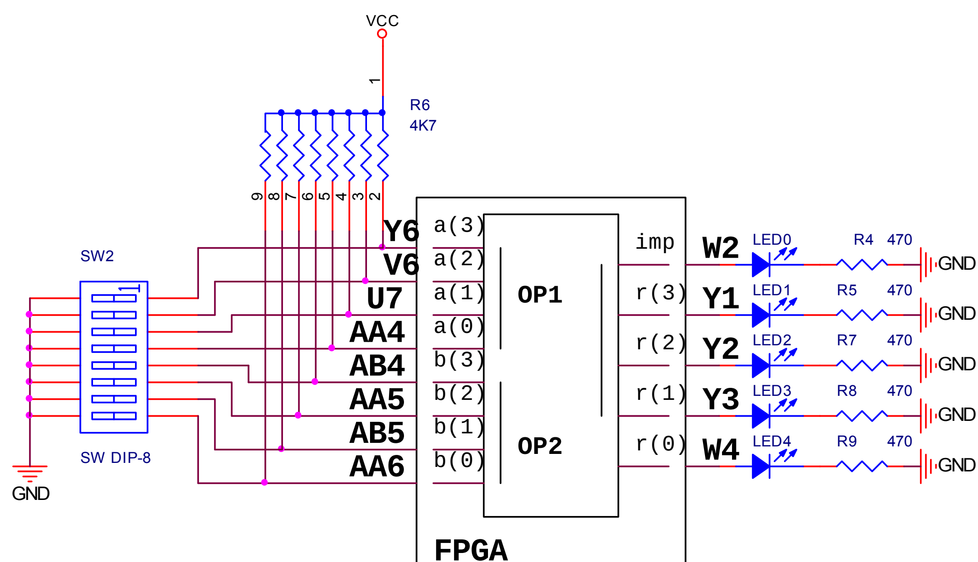
6. Să se realizeze un numărător reversibil, zecimal, cu afişare pe doi digiţi, după schema de mai jos. Numărarea către înainte se va realiza prin apăsarea tastei SW2 (UP), iar numărarea către înapoi se va face prin apăsarea tastei SW1 (DOWN). La apăsarea tastei SW3 (RST) se va realiza aducerea la 0 a numărătorului. Se va ține seama de efectul de debouncing introdus taste.



7. Să se implementeze în limbaj VHDL un numărător digital la nivel de secundă. Referința de frecvență provine de la oscilatorul local al platformei având 50MHz. Afişarea timpului se face pe doi digiți (00 - 99). După atingerea stării finale 99 se va continua cu starea inițială 00. Schema electrică a circuitului este următoarea.



8. Să se implementeze un sumator simplu pentru doi operanzi exprimați pe 4 biți utilizând următoarea schemă:



Implementarea sumatorului se va realiza prin specificații concurente. Să se realizeze o comparație între acest sumator şi cel prezentat în lucrare, din punct de vedere al vitezei de operare, respectiv al numărului blocuri logice configurabile utilizate.

Capitolul 4
Testarea modulelor digitale descrise în VHDL

Acest capitol îşi propune prezentarea mai multor modalităţi de testare a modulelor digitale descrise în limbajul VHDL. De remarcat faptul că o parte din specificaţiile utilizate în acest scop distinct (testare) nu sunt sintetizabile. Sunt introduse specificaţiile de timp şi, respectiv cele de raportare a evenimentelor, specificaţii care stau la baza testării modulelor digitale. În finalul lucrării sunt propuse probleme spre rezolvare.

1. Breviar teoretic

1.1. Specificaţii de timp

Specificaţiile de timp sunt utilizate pentru caracterizarea modulelor digitale în domeniul timp.

În VHDL sunt utilizate trei modele de caracterizare a modulelor digitale în timp:

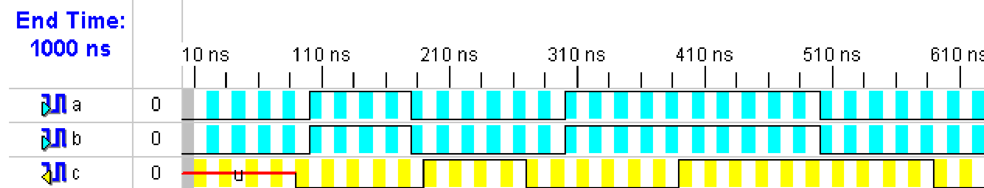
- modelul transport;
- modelul inerţial;
- modelul reject.

Modelul transport:

Codul VHDL pentru modelul de tip transport specifică timpul de propagare a unui semnal printr-un traseu electric.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity model_transport is
        port(
                a, b : in STD_LOGIC;
                c : out STD_LOGIC
          );
end model_transport;
architecture descriere of model_transport is
begin
        c <= transport (a and b) after 20ns;
end descriere;
```

Figura următoare prezintă rezultatul simulării funcționării unei porți de tip AND, cu două intrări (**a**, **b**), respectiv o ieşire (**c**). Se observă efectul de întârziere în propagarea semnalelor de la intrări către ieşire.



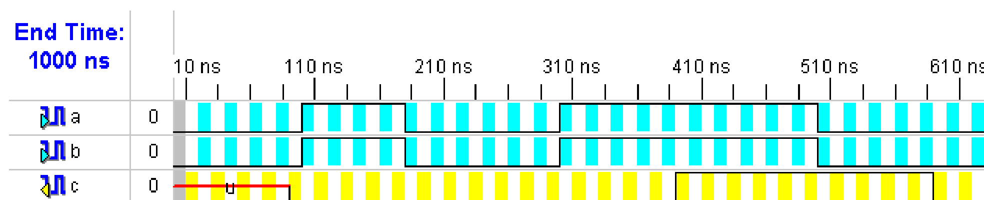
Modelul inerțial

Modelul inerțial caracterizează semnale care parcurg module digitale reale ce presupun timpi de propagare a semnalelor. Semnalele cu o durată mai mică față de perioada minimală specificată prin clauza *after* din modelul inerțial nu se propagă prin respectivul modul.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity model_inertial is
        port(
                a,b : in STD_LOGIC;
                c : out STD_LOGIC
          );
end model_inertial;

architecture descriere of model_inertial is
begin
        c <= inertial (a and b) after 100ns;
end descriere;
```



În simularea de mai sus se observă că un semnal cu durata mai mică de 100ns nu se propagă prin modulul digital. Pe de altă parte, orice semnal de ieşire din modul este întârziat cu perioada de timp precizată prin modelul inerțial (100ns).

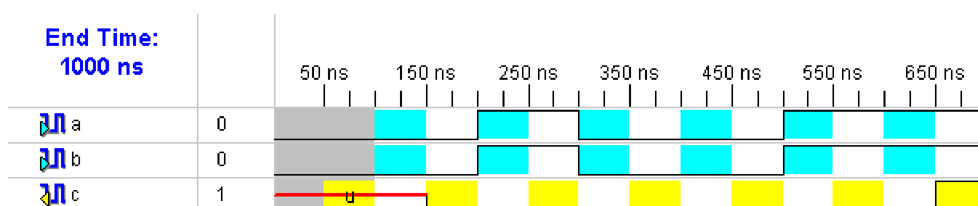
Modelul reject

Modelul reject este similar cu modelul inerțial, cu diferența dată de faptul că sunt permise doar semnale aflate în „1” logic, cu durată mai mare față de cea precizată în model.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity model_inertial is
        port(
                a,b : in STD_LOGIC;
                c : out STD_LOGIC
            );
end model_inertial;

architecture descriere of model_inertial is
begin
        c <=reject 110ns inertial (a and b) after 200ns;
end descriere;
```



În simularea de mai sus se observă faptul că semnalul aflat în „1”logic ce are o perioadă mai mică de 110ns nu se propagă prin modulul digital. Semnalul de ieşire este întârziat cu 200ns.

1.2. Specificația ASSERT

Verificarea modulelor digitale constă în obținerea rezultatelor simulării într-o formă uşor interpretabilă de către programator.

Acest lucru se poate face prin diferite metode: utilizarea facilităților interne ale simulatorului cu afişarea formelor de undă ale semnalelor de ieşire, afişarea unor liste cu valorile semnalelor de ieşire raportate la domeniul timp ale modulului digital, obținerea unui fişier extern cu rezultatele simulării acestuia sau afişarea unor mesaje scurte prin specificația **assert**.

Ultima metodă se poate adăuga simplu şi este de obicei folosită pentru a afişa un mesaj către utilizator.

O specificație de tip **assert** este alcătuită din trei elemente:

- **Assert statement** (verifică o condiție Booleană);
- **Report statement** (defineşte un mesaj ce va fi afişat când o condiție este falsă);
- **Severity statement** (informează utilizatorul despre nivelul de severitate al mesajelor).

Sintaxă:

```
ASSERT expresie_conditie
REPORT afisare_text
SEVERITY nivel_severitate;
```

În momentul când este îndeplinită o condiție în specificația ASSERT, se afişează de către consolă un text prin intermediul specificației REPORT. Suplimentar, textul afişat prezintă o notificare dată de specificația SEVERITY, ce poate avea următoarele clauze: NOTE, WARNING, ERROR, FAILURE. Apariția clauzei FAILURE conduce la oprirea simulatorului.

1.3. Testarea modulelor digitale

Testarea modulelor digitale descrise prin limbaje de tip HDL este permisă la toate nivelele de proiectare. Testarea se poate face începând cu faza de compilare, în care se verifică circuitul numai din punct de vedere logic, până la faza de implementare. În faza de implementare testarea ține seama şi de timpii de propagare.

Verificarea sistemului digital se poate realiza în toate etapele de proiectare:

- descriere în cod VHDL a modulului digital;
- sinteză;
- implementare;
- programarea structurii hardware.

Metoda de testare şi simulare în limbajul VHDL este denumită "test-bench". Test-bench-ul este un fişier VHDL prin care modulul digital (unitatea supusă testării) este verificat, interpretând semnalele de ieşire ale acestuia, ca răspuns la semnalele de intrare (stimuli). Stimulii, la rândul lor, sunt generați prin acelaşi fişier VHDL. Practic, test-bench-ul emulează mediul în care va fi plasat modulul digital, astfel încât să poată fi simulat şi analizat comportamentul acestuia.

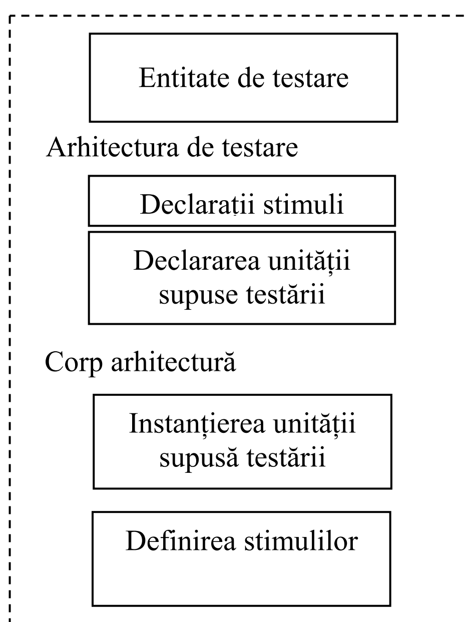
Un proiect de tip test-bench este alcătuit din următoarele elemente:

- un socket pentru unitatea supusă testării;

- un generator de stimuli (practic, un subsistem ce aplică stimuli la unitatea supusă testării, fie generați intern, fie provenind de la un fişier extern);
- submodule pentru monitorizarea răspunsurilor de către unitatea supusă testării la stimuli.

Structura unui fişier de testare VHDL

Fişierul de testare al unui modul digital este scris în cod VHDL, dar folosind alte specificații, cu entitate şi arhitectură proprii. Întotdeauna acesta are o structură particulară cu următoarele elemente constructive:



Entitatea de testare are o formă particulară, căreia îi lipsesc porturile de intrare-ieşire. Unitatea de testare nu comunică cu alte structuri exterioare.

Sintaxa entității de testare este următoarea:

```
ENTITY mod_testare IS
END mod_testare;
```

Declararea unității supuse testării se face prin introducerea unei componente corespunzătoare acesteia în zona declarativă a arhitecturii. Prin instanțierea componentei ce prezintă UUT se fac interconexiunile între unitatea de test şi unitatea supusă testării.

Stimulii sunt semnale declarate intern în zona declarativă a arhitecturii şi sunt folosiți ca porturi de intrare pentru unitatea supusă testării. Stimulii sunt definiți ca forme de undă în cadrul acestei arhitecturi prin descrieri comportamentale. De asemenea, aceştia pot fi citiți din fişiere externe fişierului de testare.

2. Aplicații

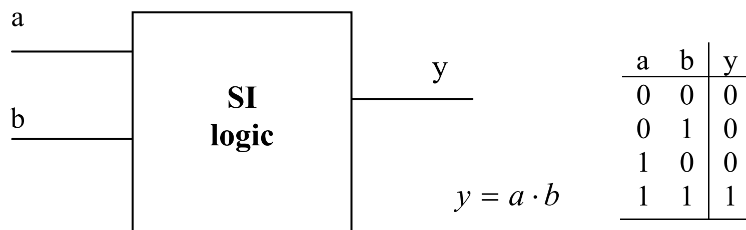
a) Folosind limbajul VHDL să se verifice funcțional o poartă logică SI prin utilizarea fişierelor de tip *test-bench* .

Soluție:

Este prezentat un exemplu foarte simplu în vederea verificării unei porți logice de tip AND. În vederea înțelegerii mecanismului de testare, în acest exemplu sunt create două fişiere VHDL:

- primul în care se face descrierea porții AND;
- al doilea în care se realizează testarea porții logice create anterior.

I. Crearea modulului digital AND



a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

În figura de mai sus este prezentată entitatea porții AND cu două intrări ce urmează a fi proiectată. Implementarea porții AND se realizează numai cu operatori logici interconectați după ecuația booleană rezultată din tabelul de adevăr al acesteia.

Se creează proiectul *testare_poarta_and*;

Se creează fişierul *poartă_and.vhd* şi se ataşează următorul cod sursă:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity poarta_xnor is
        port(
                a : in STD_LOGIC;
```

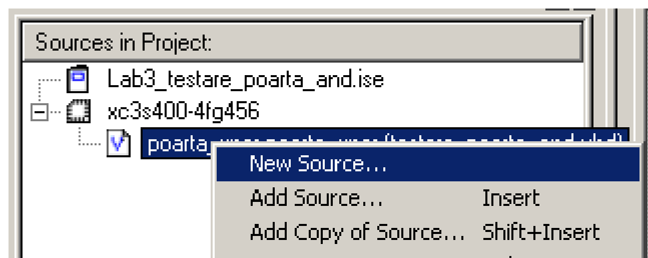
```
b : in STD_LOGIC;
y : out STD_LOGIC  );
end poarta_xnor;

architecture poarta_xnor of poarta_xnor is
begin
    y <= (not a and not b)or (a and b);
end poarta_xnor;
```

II. Scrierea fişierului de testare (test-bench file)

Antetul unui fişier de testare este generat automat de către mediul software XILINX.

Se creează un nou fişier prin selectarea butonului **New Source** după care apare fereastra **New Source**.



În acest caz se selectează **VHDL Test Bench** şi va primi numele *test_poarta_and* după care se selectează butoanele **NEXT** şi **FINISH**.

În fişierul de test este introdus următorul cod sursă:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY test_modul_and_vhd IS
END test_modul_and_vhd;

ARCHITECTURE behavior OF test_modul_and_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT poarta_xnor
    PORT(
        a : IN std_logic;
        b : IN std_logic;
```

```
            y : OUT std_logic
            );
    END COMPONENT;

    --Inputs
    SIGNAL a :  std_logic := '0';
    SIGNAL b :  std_logic := '0';

    --Outputs
    SIGNAL y :  std_logic;

BEGIN
    -- Instantiate the Unit Under Test (UUT)
    uut: poarta_xnor PORT MAP(
            a => a,
            b => b,
            y => y
    );

    tb : PROCESS
    BEGIN
            -- Wait 10 ns for global reset to finish
            wait for 10 ns;

            a <= '0';
            b <= '0';
            wait for 10 ns;

            a <= '0';
            b <= '1';
            wait for 10 ns;

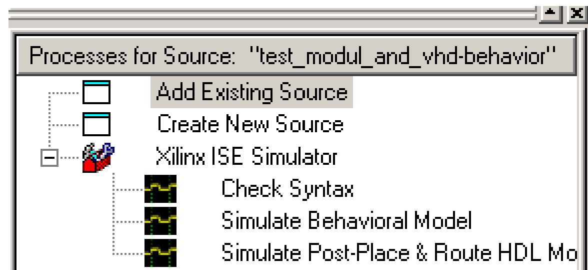
            a <= '1';
            b <= '0';
            wait for 10 ns;

            a <= '1';
            b <= '1';
            wait for 10 ns;

            a <= '0';
            b <= '0';
            wait for 10 ns;
            -- Place stimulus here
            wait; -- will wait forever
    END PROCESS;

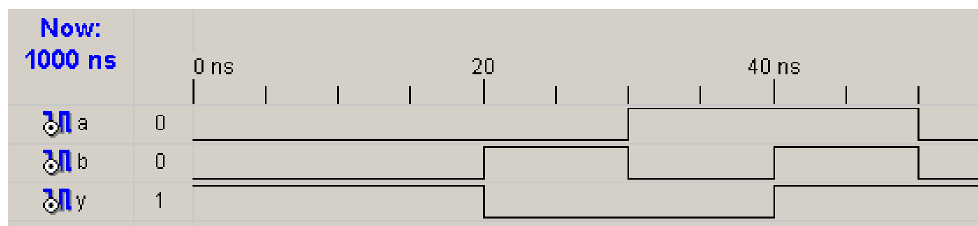
END;
```

La selectarea fisierului de test în fereastra **Module View** vor apărea următoarele procese în fereastra **Process View**:



Prin selecția procesului **Check Syntax** se verifică corectitudinea sintaxei VHDL din fişierul de test.

Simularea efectivă a proiectului se realizează prin selectarea procesului **Simulate Behaviorial Model**. După rularea acestui proces apare fereastra în care sunt trasate formele de undă a modulului digital testat.



Ca orice test-bench, programul de mai sus constă în declararea librăriei IEEE 1164, declararea entității fără a avea porturi de intrare sau ieşire şi descrierea stimulilor în cadrul arhitecturii acestuia.

În zona declarativă a arhitecturii fişierului de test, modulul digital *mod_and* este declarat ca o componentă. Practic este ca un circuit integrat care este plasat într-un socket de test prin procesul de instanțiere.

Sunt declarate semnalele interne **a**, **b** şi **c**. Acestea sunt semnale locale utilizate ca intrări în unitatea supusă testări, respectiv ca ieşiri pentru observarea comportamentului acesteia.

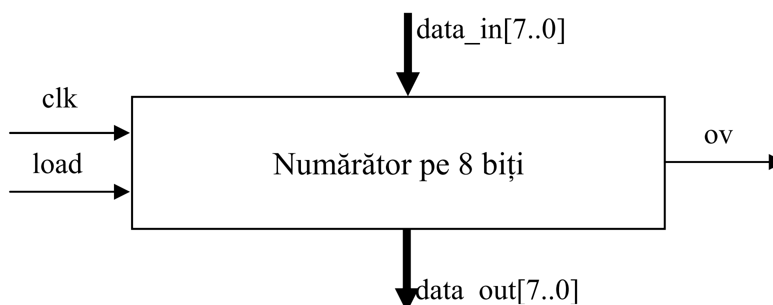
În cadrul procesului este descris comportamentul semnalelor de intrare în funcție de timp. Acest proces a fost descris fără a folosi o listă de senzitivități. În acesta se utilizează clauza WAIT pentru a specifica perioada la care apare o modificare pe semnalele de intrare ale modului digital.

b) Să se realizeze un numărător de tip BCD pe 8 biți, cu încărcare a valorii inițiale şi antenționarea cazului în care se trece de la valoarea maxima (99) la valoarea minimă (00).

Soluție:

Realizarea unui numărător pe 8 biți este o aplicație tipică în limbajul VHDL. Numărătorul prezintă, un semnal **load** prin care se face inițializarea acesteia cu o valoare dată pe portul **data_in**.

Porturile de intrare/ieşire ale numărătorului digital sunt date în figura de mai jos:



Modulul digital se incrementează după semnalul de ceas **clk** pe frontul pozitiv;

Încărcarea numărătorului se realizează atunci când semnalul **load** este în 1 logic;

După trecerea semnalului **load** în 0 logic, incrementarea începe la următorul front pozitiv al semnalului de ceas;

Semnalul **ov** (overflow) este în 0 logic atât timp numărul incrementat nu a ajuns la valoarea 0x99. Setarea acestuia în 0 logic se face pe următorul front de ceas;

În vederea testării acestui modul digital vor fi parcurşi următorii paşi:

Se creează un proiect cu numele *num_zec*;

Se creează un fişier VHDL cu numele *num_zec.vhd*;

Se introduce următorul program sursă în VHDL care descrie funcționarea modulului digital.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity num_zec is
   Port ( clk : in std_logic;
          load : in std_logic;
          ov : out std_logic;
          data_in : in std_logic_vector(7 downto 0);
          data_out : out std_logic_vector(7 downto 0));
end num_zec;

architecture Behavioral of num_zec is
begin
        process(clk)
                variable temp: std_logic_vector(7 downto 0):=(others => '0');
        begin
          if (clk'event and clk='1') then
                        ov <= '0';
                        if load = '1' then temp := data_in;
                        else    temp := temp + '1';
                                if temp(3 downto 0)>= "1010" then
                                        temp:= temp+ x"06";
                                end if;

                                if temp(7 downto 4)>= "1010" then
                                        temp:= (others => '0');
                                        ov <='1';
                                end if;
                        end if;

                end if;
                        data_out <= temp;
        end process;

end Behavioral;
```

Numărătorul digital este implementat printr-un singur singur process sincron după semnalul **clk**. În cadrul procesului este declarată o variabilă numită **temp**. Această variabilă este incrementată la fiecare front pozitiv al semnalului de ceas după care este testat octetul cel mai puțin semnificativ al acesteia daca a ajuns la valoarea 0x0A iar pentru octetul cel mai semnificativ se aplică un test similar. Aceste două teste permit numărarea în cod BCD.

În finalul procesului, semnalul de ieşire **data_out** primeşte valoarea variabilei **temp**.

În cadrul acestui proces este testat şi semnalul **load**, sincron după semnalul de ceas, prin care este inițializată variabila **temp** cu valoarea semnalului de intrare **data_in**.

Realizarea fişierului de test

În acelaşi proiect se realizează un nou fişier VHDL cu numele *test_num_zec.vhd* în care este plasat următorul program:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY test_num_zec_vhd IS
	GENERIC(perioada : time := 100ns);
END test_num_zec_vhd;

ARCHITECTURE behavior OF test_num_zec_vhd IS

	-- Component Declaration for the Unit Under Test (UUT)
	COMPONENT num_zec
	PORT(
		clk : IN std_logic;
		load : IN std_logic;
		data_in : IN std_logic_vector(7 downto 0);
		ov : OUT std_logic;
		data_out : OUT std_logic_vector(7 downto 0)
		);
	END COMPONENT;

	--Inputs
	SIGNAL clk :  std_logic := '0';
	SIGNAL load :  std_logic := '0';
	SIGNAL data_in :  std_logic_vector(7 downto 0) := (others=>'0');

	--Outputs
	SIGNAL ov :  std_logic;
	SIGNAL data_out :  std_logic_vector(7 downto 0);

BEGIN

	-- Instantiate the Unit Under Test (UUT)
	uut: num_zec PORT MAP(
		clk => clk,
		load => load,
		ov => ov,
		data_in => data_in,
		data_out => data_out
	);
```

```
process
	variable reg_ceas: STD_LOGIC:='1';
begin
	clk <= reg_ceas;
	reg_ceas:= not reg_ceas;
	wait for perioada / 2;
end process;


tb : PROCESS
BEGIN

	-- Wait 100 ns for global reset to finish
	wait for 100 ns;

	data_in<=x"53";
	wait for perioada;

	load<='1';
	wait for 2*perioada;

	assert (load /='1')
	report "Numaratorul se incarca cu o valoare initiala"
severity NOTE;

	load<='0';
	wait for perioada;

	for i in 0 to 80 loop
		assert (ov /='1')
		report "Apare depasire" severity NOTE;
		wait for perioada;
	end loop;

	wait; -- will wait forever
END PROCESS;

END;
```

În vederea reducerii drastice a mărimii codului sursă au fost introduse facilități ale limbajului VHDL. De exemplu bucle de tip **for-loop**, constante iar pentru atenționarea programatorului în timpul simulării este utilizată specificația **assert**.

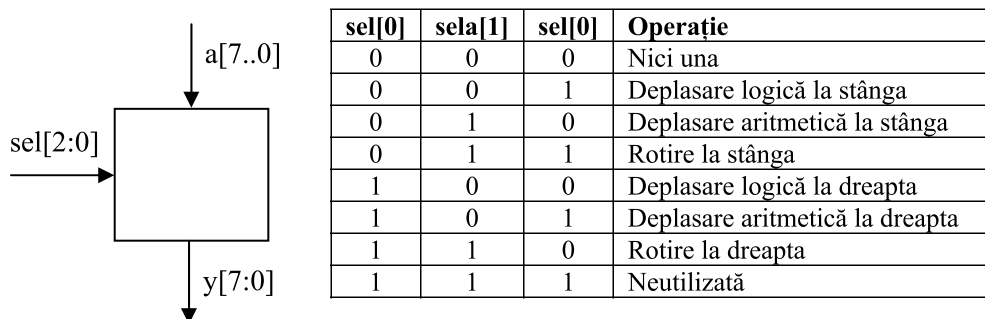
Utilizarea proceselor multiple formează un cod sursă VHDL structurat şi totodată pentru programator uşor de controlat.

În primul proces este generat semnalul de ceas cu o perioadă de 100ns. Timpul pentru o perioadă este dat prin constanta *perioada*. Variabila reg_ceas este utilizată pe post de registru pentru memorarea stării semnalului de ceas.

În procesul al doilea este descris testul propriu-zis al numărătorului zecimal. Acesta este inițilizat prin semnalul ***data_in*** cu valoarea hexazecimală 53. După 100ns se realizează inițializarea încărcării datei paralele prin setarea variabilei ***load*** în 1 logic.

În final se trece la generarea a mai multor perioade de ceas pentru testarea modulului digital.

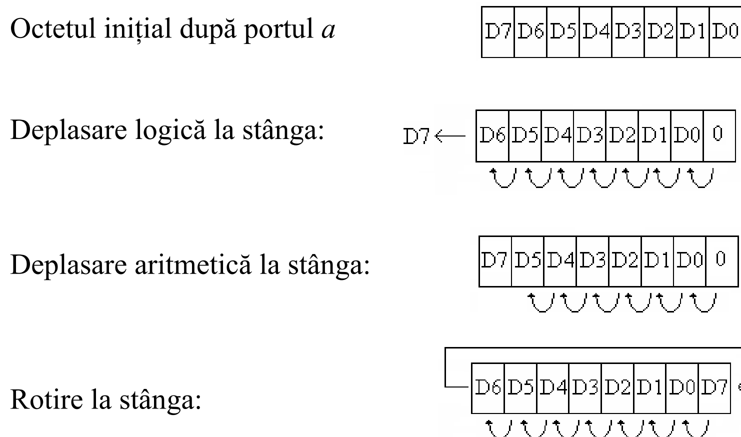
c) Să se realizeze un fisier de tip test-bench care testează un modul digital de tip registru cu entitatea dată în figura de mai jos şi care realizează operațiile de deplasare stânga, dreapta în funcție de portul de selecție ca în tabelul alăturat.



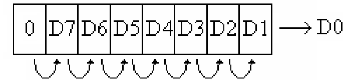
sel[0]	sela[1]	sel[0]	Operație
0	0	0	Nici una
0	0	1	Deplasare logică la stânga
0	1	0	Deplasare aritmetică la stânga
0	1	1	Rotire la stânga
1	0	0	Deplasare logică la dreapta
1	0	1	Deplasare aritmetică la dreapta
1	1	0	Rotire la dreapta
1	1	1	Neutilizată

Soluție:

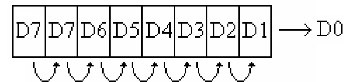
În primul rând sunt date operațiile de deplasare din tabelul de mai sus aplicate de modulul digital asupra octetului preluat după portul de intrare al acestuia.



Deplasare logică la dreapta:



Deplasare aritmetică la dreapta:



Rotire la dreapta:



În cadru acestui elemplu, pentru o întelegere mai bună, este creat în afara fişierului de test si modulul de deplasare. Ordinea operațiilor este următoarea:

Se creează un proiect cu numele *reg_depls*;

Se creează un fişier VHDL cu numele *reg_depls*;

Se introduce următorul program sursă în VHDL care descrie funcționarea registrului **reg_depls**.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity reg_depls is
	port(
		a : in STD_LOGIC_VECTOR(7 downto 0);
		sel : in STD_LOGIC_VECTOR(2 downto 0);
		y : out STD_LOGIC_VECTOR(7 downto 0)
		);
end reg_depls;

architecture reg_depls of reg_depls is
begin

	with sel select
	y<= a when "000",
		a(6 downto 0) & '0'		when "001",
		a(7) & a(5 downto 0) & '0'	when "010",
		a(6 downto 0)& a(7)		when "011",
		'0' & a(7 downto 1)		when "100",
		a(7) & a(7 downto 1)		when "101",
		a(0) & a(7 downto 1)		when "110",
		(others => 'Z')			when others;

end reg_depls;
```

Alegerea operaţiei de deplasare ce se aplică asupra portului de intrare **sel** se realizează în cadrul programului VHDL prin specificaţia de selecţie concurentă *with-select-when*.

Pentru simularea programului de mai sus se va utiliza o nouă metodă în care formele de undă ale stimulilor se găsesc într-un fişier extern. Se doreşte ca şi rezultatele obţinute în urma simulării să fie plasate într-un fişier de ieşire.

Fişierul de intrare care conţine formele de undă a stimulilor se va numi *data_in.dat* iar fişierul de ieşire se va numi *data_out. dat*.

În acelaşi proiect în care a fost introdus reg_depls.vhd se realizează un nou fişier VHDL (fişierul de test) cu numele *reg_depls_test.vhd* conţinând următorul program sursă:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use STD.textio.all;

entity reg_deplasare_test is
end reg_deplasare_test;


architecture reg_deplasare_test of reg_deplasare_test is
        component reg_depls
                port(
                        a : in STD_LOGIC_VECTOR(7 downto 0);
                        sel : in STD_LOGIC_VECTOR(2 downto 0);
                        y : out STD_LOGIC_VECTOR(7 downto 0)
                        );
        end component;

        signal          a_tb : STD_LOGIC_VECTOR(7 downto 0);
        signal          sel_tb : STD_LOGIC_VECTOR(2 downto 0);
        signal          y_tb : STD_LOGIC_VECTOR(7 downto 0);

begin

        U1: reg_depls port map(a_tb,sel_tb,y_tb);

        process

        file rd_fis: text open READ_MODE is "data_in.dat";
        file wr_fis: text open WRITE_MODE is "data_out.dat";

                procedure citeste_fis is
                variable l1 : line;
                variable sel_var :bit_vector(2 downto 0);
```

```
                variable a_var : bit_vector(7 downto 0);
        begin
                if ((not (endfile(rd_fis)))) then
                        readline(rd_fis, l1);
                        read(l1,sel_var);
                        sel_tb <= to_StdLogicVector(sel_var);
                        read(l1, a_var);
                        a_tb <= to_StdLogicVector(a_var);
end if;
        end citeste_fis;
        procedure scrie_fis is
                variable z1 : line;
                variable y_var : bit_vector(7 downto 0);
        begin
                y_var := to_bitvector(y_tb);
                write (z1, y_var);
                writeline(wr_fis, z1);
        end scrie_fis;

begin
        for i in 1 to 4 loop
                citeste_fis;
                wait for 50 ns;
                scrie_fis;
        end loop;

        wait;
end process;

end reg_deplasare_test;
```

Modulul digital este simulat cu ajutorul unui program test-bench în care nu a fost necesară generarea stimulilor de intrare ca în exemplul anterior. Aceştia au fost citiți dintr-un fişier de intrare *data_in.dat*. De asemenea, rezultatele au fost salvate într-un fişier de ieşire după care pot fi citite de către utilizator separat pentru a identifica eventualele erori apărute pe parcursul simulării, în cazul în care acestea există.

În primul rând a fost declarată librăria **STD** cu pachetul **textio** pentru lucrul cu fişiere. S-au creat două proceduri *citeste_fis* şi *scrie_fiş* prin care se realizează citirea din fişierul de intrare şi scrierea în fişierul de ieşire.

Fişierul de intrare se creează urmând paşii:

- se deschide un fişier nou de *tip .txt*;
- se redenumeşte în *data_in.dat*;
- este completat cu datele de mai jos cu ajutorul unui editor de text.

```
000 10101010
001 11110000
010 11001100
011 01010101
```

În procedura *citeşte_fis* sunt preluaţi stimulii *sel_tb* şi *a_tb* pentru a fi utilizaţi ca date de intrare pentru modulul *reg_depls*. Din cauza faptului că librăria STD face parte din standardul 1173, acesta nu are definit tipul std_logic şi este necesară o funcţie de conversie din tipul *bit* în tipul *std_logic*. În cadrul procedurii, fişierul este citit linie cu linie (fiecare linie conţinând câte 11 biţi). Primii trei biţi reprezintă semnalul de selecţie **sel** iar următorii 8 reprezintă portul de intrare **a**. Citirea celor două valori se face secvenţial.

Prin procedura *scrie_fiş* se realizează operaţia inversă celei anterioare. În fişierul *data_out.dat* este scris doar semnalul de ieşire y din modulul de deplasare. În acest caz este necesară conversia din tipul *std_logic* în binar pentru ca datele să poată fi salvate în fişier.

Corpul procedurii de simulare este format dintr-o specificaţie *for* prin care sunt realizaţi 4 ciclii de simulare (reprezintă numărul stimulilor din fişierul de intrare). În cadrul acestuia se face o citire a unui set de stimuli după care se aşteaptă o perioadă de 50ns şi apoi se va salva răspunsul modulului de deplasare în fişierul de ieşire.

Pentru cazul în care nu este dorită citirea tuturor stimulilor din fişierul de intrare, de exemplu, în acest caz să fie citit din fişierul de intrare numai semnalul **a** iar semnalul **sel** să fie generat prin program se modifică corpul arhitecturii anterioare după cum este dat în următoarele linii de cod:

```
process

file rd_fis: text open READ_MODE is "data_in.dat";
file wr_fis: text open WRITE_MODE is "data_out.dat";

        procedure citeste_fis is
        variable l1 : line;
        variable sel_var :bit_vector(2 downto 0);
                variable a_var : bit_vector(7 downto 0);
        begin
                if ((not (endfile(rd_fis)))) then
                        readline(rd_fis, l1);
                        read(l1, a_var);
                        a_tb <= to_StdLogicVector(a_var);
        end if;
        end citeste_fis;


        procedure scrie_fis is
```

```
			variable z1 : line;
			variable y_var : bit_vector(7 downto 0);
		begin
			y_var := to_bitvector(y_tb);
			write (z1, y_var);
			writeline(wr_fis, z1);
		end scrie_fis;

	begin
		for i in 1 to 4 loop
			citeste_fis;

			wait for 50 ns;
			scrie_fis;
		end loop;

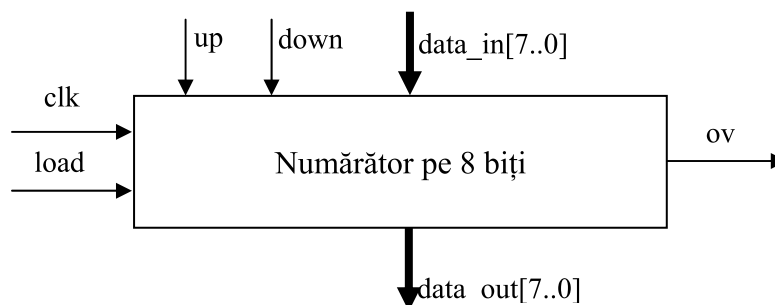
		wait;
	end process;

	process
	variable sel_var: std_logic_vector(2 downto 0):="000";
	begin
			sel_var := sel_var+1;
			sel_tb <= sel_var;
			wait for 50 ns;
	end process;
```

Este creat în plus un proces în care este incrementat semnalul **sel** la fiecare 50ns. În cadrul procedurii de citire a datelor din fişierul de intrare sunt scoase liniile în care se realiza citirea semnalului **sel**.

3. Exerciții

1. Să se realizeze simulările funcționale prin intermediul fişierelor de tip test-bench a exemplelor descrise în cadrul laboratorului;
2. Realizați în limbajul VHDL un numărător BCD pe 8 biți cu porturile date în figura de mai jos:



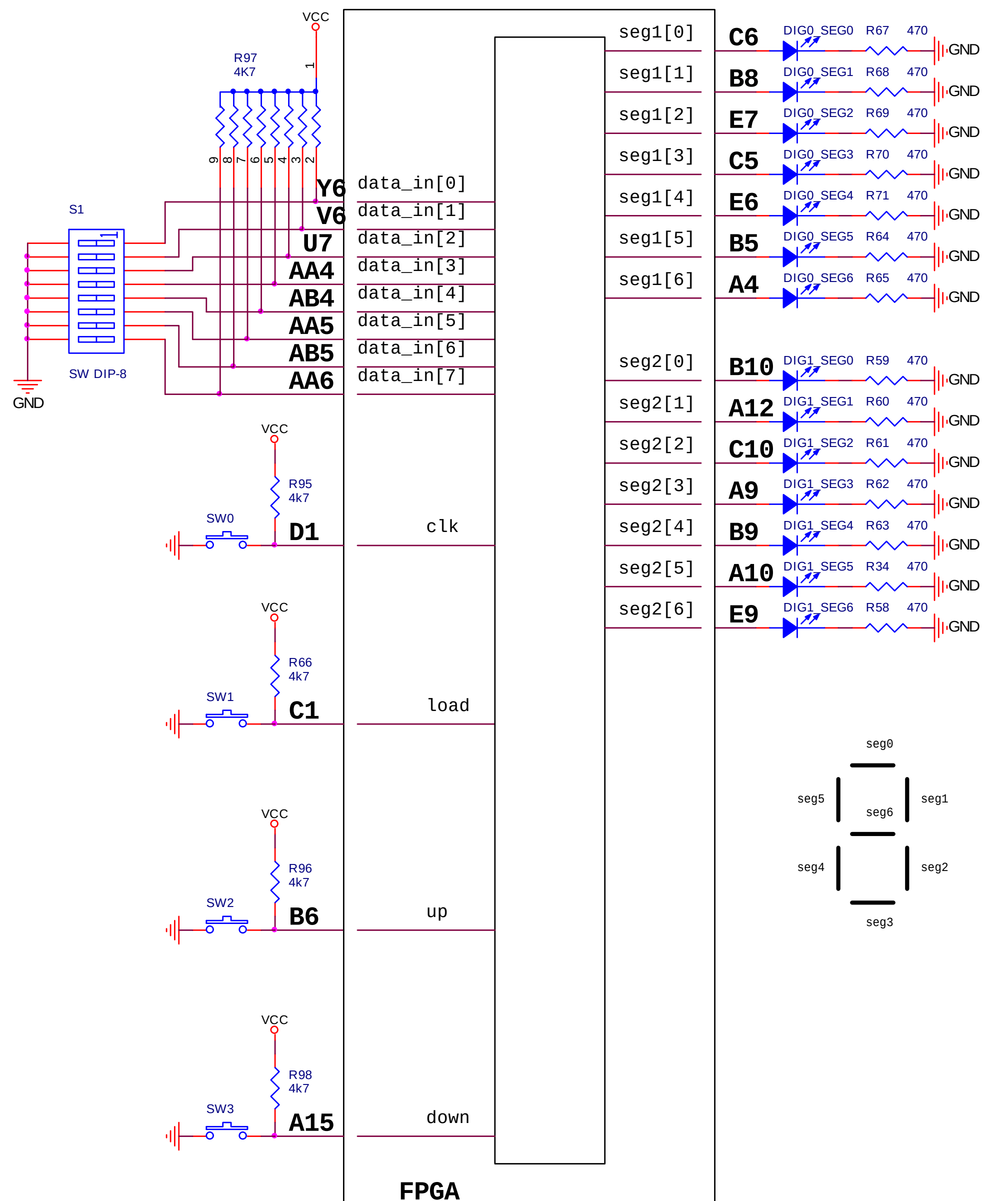
După crearea codului sursă a modulului digital să se realizeze fişierul de test pentru acesta prin care să se verifice în totalitate funcționalitatea acestuia.

În final să i se ataşeze un modul de decodare binar – 7 segmente pentru afişarea valorii numerice pe doi digiți după sistemul de laborator.

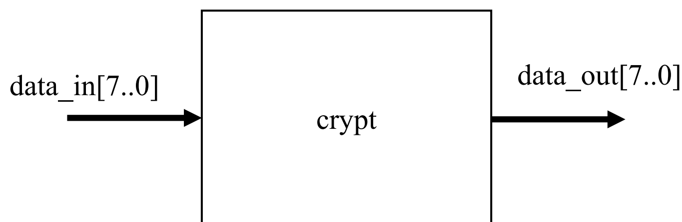
Valoarea inițială (portul **data_in**)se va încărca de la cu ajutorul switch-urilor iar restul semnalelor de intrare (porturile **clk, load, up** şi **down**) vor fi atribuite butoanelor de pe machetă.

În pus, pentru porturile **clk, load, up şi down** va fi atribuit câte un modul de tip debouncing.

Schema electrică de interconectare este dată în figura următoare:


VCC
R97
4K7
S1
SW DIP-8
GND
Y6
V6
U7
AA4
AB4
AA5
AB5
AA6
data_in[0]
data_in[1]
data_in[2]
data_in[3]
data_in[4]
data_in[5]
data_in[6]
data_in[7]
R95
4k7
SW0
D1
clk
R66
4k7
SW1
C1
load
R96
4k7
SW2
B6
up
R98
4k7
SW3
A15
down
FPGA
seg1[0]
seg1[1]
seg1[2]
seg1[3]
seg1[4]
seg1[5]
seg1[6]
seg2[0]
seg2[1]
seg2[2]
seg2[3]
seg2[4]
seg2[5]
seg2[6]
C6 DIG0_SEG0 R67 470 GND
B8 DIG0_SEG1 R68 470 GND
E7 DIG0_SEG2 R69 470 GND
C5 DIG0_SEG3 R70 470 GND
E6 DIG0_SEG4 R71 470 GND
B5 DIG0_SEG5 R64 470 GND
A4 DIG0_SEG6 R65 470 GND
B10 DIG1_SEG0 R59 470 GND
A12 DIG1_SEG1 R60 470 GND
C10 DIG1_SEG2 R61 470 GND
A9 DIG1_SEG3 R62 470 GND
B9 DIG1_SEG4 R63 470 GND
A10 DIG1_SEG5 R34 470 GND
E9 DIG1_SEG6 R58 470 GND
seg0
seg5
seg6
seg1
seg4
seg2
seg3

3. Descrieți în limbajul VHDL un sistem pentru criptarea unui flux de numere binare reprezentate pe câte 8 biți cu următoarea configurație:



Relația după care se realizează operația de criptare are forma:

$$data_out[7..0] = data_in[0..7]\ XOR\ \ 0x47$$

Realizați codul VHDL pentru modulul de mai sus caracterizat de ecuația de mai sus după care creați un fişier de tip test-bench prin care realizați o verificare funcțională a acestuia.

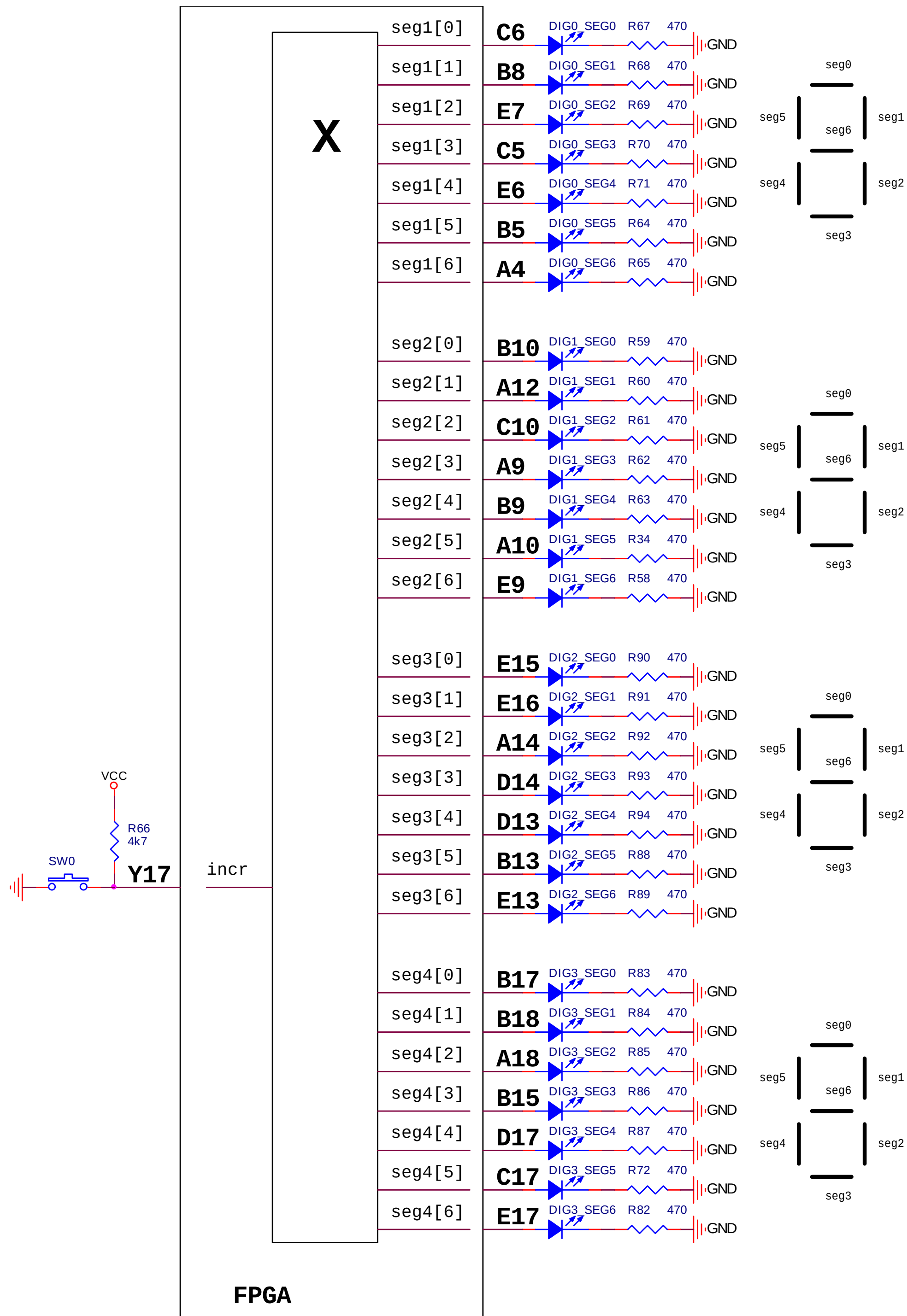
Programul de test va fi conceput astfel încât stimulii de intrare pentru semnalul **data_in** să fie preluați dintr-un fişier cu denumirea *data_in.dat* iar rezultatul simulării să fie salvat într-un fişier cu denumirea *data_out.dat*.

4. Realizați un sistem asemănător cu cel anterior dar prin care se realizează decriptarea datelor criptate cu relația de mai sus. Determinați relația de decodare după care implementați programul VHDL şi testarea modulului rezultat să fie la fel cu cea de la modulul anterior.

5. Să se realizeze în limbaj VHDL un generator de numere binare după algoritmul lui Fibonacii. Afişarea numerelor binare să se realizeze pe 4 digiți în format zecimal iar incrementarea acestora să se realizeze prin apăsarea tastei **sw0**. La apăsarea tastei **sw0** se ține cont de efectul de debouncing.

Simulați modulul generator Fibonacii printr-un fişier de tip test-bench iar rezultatele obținute să fie salvate într-un fişier care va fi folosit pentru a verifica dacă valorile numerice afişate pe digiții sistemului reconfigurabil sunt aceleaşi.

Schema electrică de implementare a modulului digital este dată în figura de mai jos.



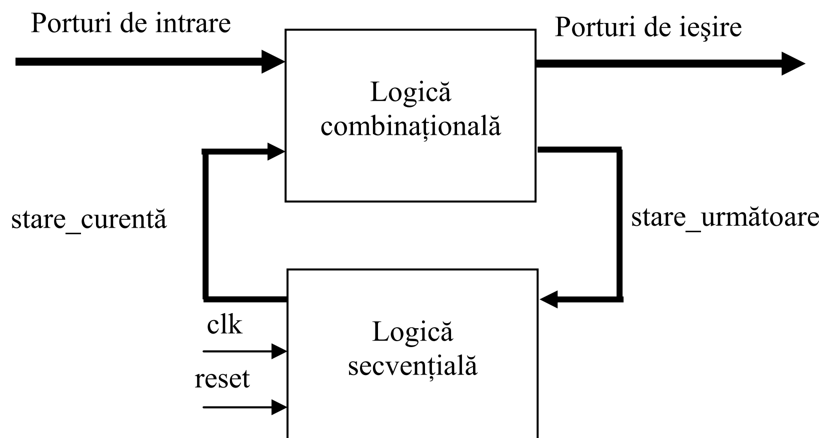
X
FPGA
incr
VCC
R66
4k7
SW0
Y17
seg1[0]
seg1[1]
seg1[2]
seg1[3]
seg1[4]
seg1[5]
seg1[6]
C6
B8
E7
C5
E6
B5
A4
seg2[0]
seg2[1]
seg2[2]
seg2[3]
seg2[4]
seg2[5]
seg2[6]
B10
A12
C10
A9
B9
A10
E9
seg3[0]
seg3[1]
seg3[2]
seg3[3]
seg3[4]
seg3[5]
seg3[6]
E15
E16
A14
D14
D13
B13
E13
seg4[0]
seg4[1]
seg4[2]
seg4[3]
seg4[4]
seg4[5]
seg4[6]
B17
B18
A18
B15
D17
C17
E17
GND
seg0
seg1
seg2
seg3
seg4
seg5
seg6

Capitolul 5
Proiectarea automatelor secvențiale cu limbajul VHDL

Acest capitol își propune sinteza și implementarea modulelor digitale în structuri hardware de tip FPGA bazate pe automate cu stări finite descrise în limbajul VHDL. Sunt prezentate aplicații de proiectare a automatelor secvențiale după modele standard recomandate în programarea cu limbajul VHDL. În finalul lucrării sunt propuse probleme spre rezolvare.

1. Breviar teoretic

Automatele cu stări finite constituie o tehnică particulară în modelarea circuitelor cu logică secvențială. În figura de mai jos este prezentată forma generală a unui automat cu stări finite privit din punctul de vedere al descrierii cu un limbaj de descriere hardware. Modulul –logică secvențială- realizează logica pentru starea curentă și este format, în general, din bistabili. Modulul –logică combinațională- realizează logica pentru starea următoare, respectiv logica pentru semnalele de ieșire care este format din elemente combinaționale.



Porturile de intrare/ieșire sunt conectate la blocul de logică combinațională. În general, în proiectarea hardware, blocul de logică secvențială este sincron cu semnalul de ceas. Dacă este activ semnalul **reset**, starea curentă va fi cea inițială.

Dacă portul de ieşire a automatului secvențial depinde de starea prezentă şi de valoarea logică a porturilor de intrare, atunci acesta este de tip Mealy.

Automatul secvențial este de tip Moore atunci când portul de ieşire depinde numai de starea prezentă a acestuia.

În limbajul VHDL, cele două blocuri (combinațional, respectiv secvențial) pot fi realizate prin intermediul unuia sau mai multor procese. Blocul combinațional va fi proiectat numai cu specificații concurente prin descriere de tip flux de date sau printr-un proces care descrie un circuit combinațional. Al doilea bloc este implementat printr-un proces ce descrie un circuit secvențial, sincron după semnalul de ceas.

Pentru implementarea automatelor cu stări finite sunt utilizate mai multe metode de descriere a acestora. Pot fi descrise doar printr-un proces, două procese sau trei procese.

În acest laborator este utilizată metoda de descriere a unui automat cu stări finite prin trei procese.

1.1. Implementarea blocului ce asigura starea urmatoare

Blocul secvențial prezintă un proces senzitiv după două semnale (**clk** şi **reset**). Acest bloc, sincron cu semnalul de ceas şi asincron cu semnalul de reset, asigură starea prezentă. La reset, starea prezentă va fi întotdeauna starea inițială a automatului. Procesul de mai jos are avantajul că ocupă un număr mic de bistabili. Programul VHDL care descrie acest bloc, are în general forma următoare:

```
process(reset, clock)
begin
        if (reset ='1') then
                stare_prezenta <= stare_initiala;
        elsif (clk'event and clk = '1') then
                stare_prezenta <= stare_urmatoare;
        endif;
end process;
```

1.2. Implementarea blocului ce asigura starea curenta

Blocul combinațional poate fi descris prin intermediul a două procese (logica de ieşire şi logica ce asigură starea următoare) sau integral cu specificații concurente. Modelul cel mai uşor de realizat utilizează un proces în care se face selecția stărilor şi un al doilea proces în care se realizează selecția ieşirilor în funcție de stări. Ambele selecții, în general, se fac prin specificația CASE sau prin selecție condițională cu specificația IF THEN. Programul generic VHDL pentru implementarea blocului combinațional de selecție al stărilor este următorul:

```
process(port_intrare, stare_prezenta)
begin
        case stare_prezenta is
                when stare0 =>
                        if (intrare = … ) then
                                port_iesire <= <valoare>;
                                stare_urmatoare <= stare1;
                        else …
                        end if;
                when stare1 =>
                        if (intrare = … ) then
                                port_iesire <= <valoare>;
                                stare_urmatoare <= stare2;
                        else …
                        end if;
                when stare2 =>
                        if (intrare = … ) then
                                port_iesire <= <valoare>;
                                stare_urmatoare <= stare3;
                        else …
                        end if;
                …
                when others null;
        end case;
end process;
```

1.3. Implementarea blocului de decodare a porturilor de ieşire

Procesul de decodare logică a porturilor de ieşire în funcție de starea prezentă se realizează după două modele:

- **Modelul Mealy** în care se ține seama de starea prezentă şi de intrări

```
process(port_intrare, stare_prezenta)
begin
        case stare_prezenta is
                when stare0 =>
                        if (intrare = … ) then
                                port_iesire <= <valoare>;
                        else …
                        end if;
                when stare1 =>
                        if (intrare = … ) then
                                port_iesire <= <valoare>;
                        else …
                        end if;
```

```
                when stare2 =>
                        if (intrare = … ) then
                                port_iesire <= <valoare>;
                        else …
                        end if;
                …
                when others null;
        end case;
end process;
```

- **Modelul Moore** în care se ține cont numai de starea prezentă:

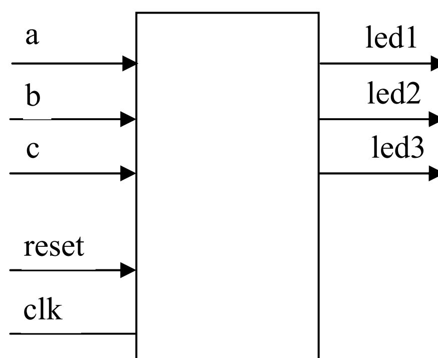
```
process(stare_prezenta)
begin
        case stare_prezenta is
                when stare0 =>          port_iesire <= <valoare>;
                when stare1 =>          port_iesire <= <valoare>;
                when stare2 =:          port_iesire <= <valoare>;
                …
                when others null;
        end case;
end process;
```

2. Aplicații

a) Să se descrie în cod VHDL un detector care identifică o secvență de apăsare consecutivă a trei taste. Fiecare apăsare corectă este indicată prin aprinderea unui led. În cazul în care a fost apăsată greşit o tastă se sting toate ledurile şi trebuie reluată secvența de apăsări a tastelor.

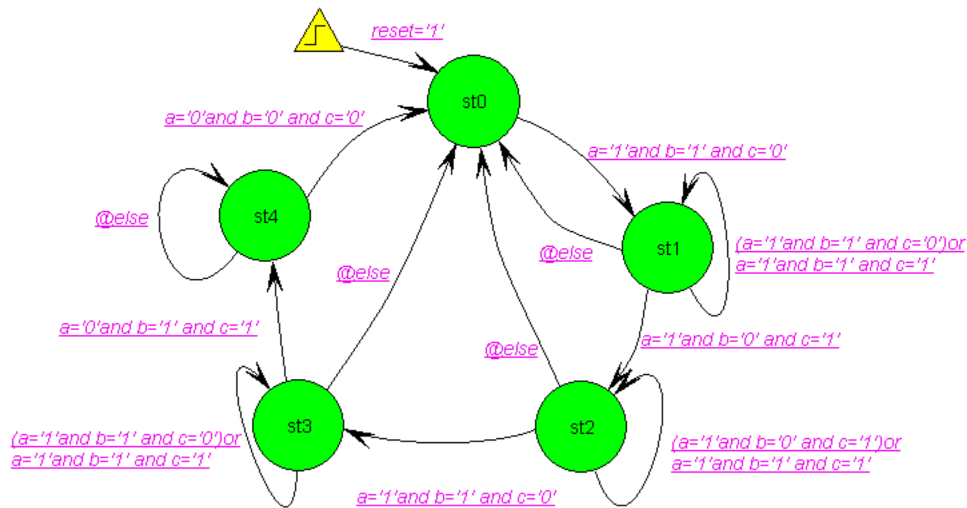
Porturile de intrare corespunzătoare tastelor sunt a, b şi c. Porturile de ieşire corespunzătoare ledurilor sunt notate cu led1, led2, led3 şi led4.

Reprezentarea grafică a entității modulului digital este dată în figura de mai jos.



Soluţie

Pentru implementarea acestui automat secvenţial a fost realizată următorul graf de funcţionare



Dacă este apăsată tasta **c,** din starea **st0** se trece la **st1** şi se aprinde primul led. Rămâne în această stare atâta timp cât este apăsată această tastă sau dacă nu este apăsată nici-o tastă. Dacă este apăsată tasta **b**, se trece la starea următoare aprinzându-se încă un led, altfel se trece în starea **st0** şi se sting toate ledurile. Acelaşi lucru se întâmplă şi pentru stările **st2** şi **st3**. Starea **st4** este cea finală în care se aprind toate ledurile. Se iese din această stare numai atunci când sunt apăsate toate butoanele.

Dacă se doreşte iniţializarea sistemului din orice stare se apasă tasta **reset**.

Programul în VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity cifru_taste is
   Port ( a : in std_logic;
          b : in std_logic;
          c : in std_logic;
          clk : in std_logic;
          reset : in std_logic;
          led1 : out std_logic;
          led2 : out std_logic;
          led3 : out std_logic;
```

```
        led4 : out std_logic);
end cifru_taste;

architecture Behavioral of cifru_taste is
    type tip_stare is (st0, st1, st2, st3, st4);
    signal st_prez, st_urm: tip_stare;

    signal data_in: std_logic_vector(2 downto 0);
    signal data_out: std_logic_vector(3 downto 0);

begin

    process(reset, clk)
    begin
            if (reset='0')then
                    st_prez <= st0;
            elsif (clk'event and clk='1')then
                    st_prez <= st_urm;
            end if;
    end process;

    process(st_prez)
    begin
            case(st_prez) is
                    when st0 => data_out <="0000";
                    when st1 => data_out <="0001";
                    when st2 => data_out <="0011";
                    when st3 => data_out <="0111";
                    when st4 => data_out <="1111";
                    when others => null;
            end case;
    end process;

    process(st_prez, data_in)
    begin
            case (st_prez) is
                    when st0 =>
                            if (data_in="110") then
                                    st_urm <= st1;
                            else
                                    st_urm <= st0;
                            end if;

                    when st1 =>
                            if (data_in="101") then
                                    st_urm <= st2;
                            elsif (data_in="111" or data_in="110") then
                                    st_urm <= st1;
```

```
                    else
                            st_urm <= st0;
                    end if;

            when st2 =>
                    if (data_in="110") then
                            st_urm <= st3;
                    elsif (data_in="111" or data_in="101") then
                            st_urm <= st2;
                    else
                            st_urm <= st0;
                    end if;

            when st3 =>
                    if (data_in="011") then
                            st_urm <= st4;
                    elsif (data_in="111" or data_in="110") then
                            st_urm <= st3;
                    else
                            st_urm <= st0;
                    end if;

            when st4 =>
                    if (data_in="000") then
                            st_urm <= st0;
                    else
                            st_urm <= st4;
                    end if;
            when others => null;
        end case;
    end process;

    data_in <= c & b & a;

    led1 <= data_out(0);
    led2 <= data_out(1);
    led3 <= data_out(2);
    led4 <= data_out(3);

end Behavioral;
```

Acest program este format din trei procese:

Primul proces este standard şi realizează tranziţia de la *starea următoare la starea prezentă* pe frontul pozitiv de ceas.

În al doilea proces se realizează *decodarea portului de ieşire în funcţie de starea prezentă*. Se observă că portul de ieşire este **data_out** şi nu led1 … led3. Practic semnalele de ieşire care controlează ledurile sunt comandate cu un singur

semnal de tip vector (**data_out**) iar repartiția către leduri se face prin următoarele linii de cod:

```
led1 <= data_out(0);
led2 <= data_out(1);
led3 <= data_out(2);
led4 <= data_out(3);
```

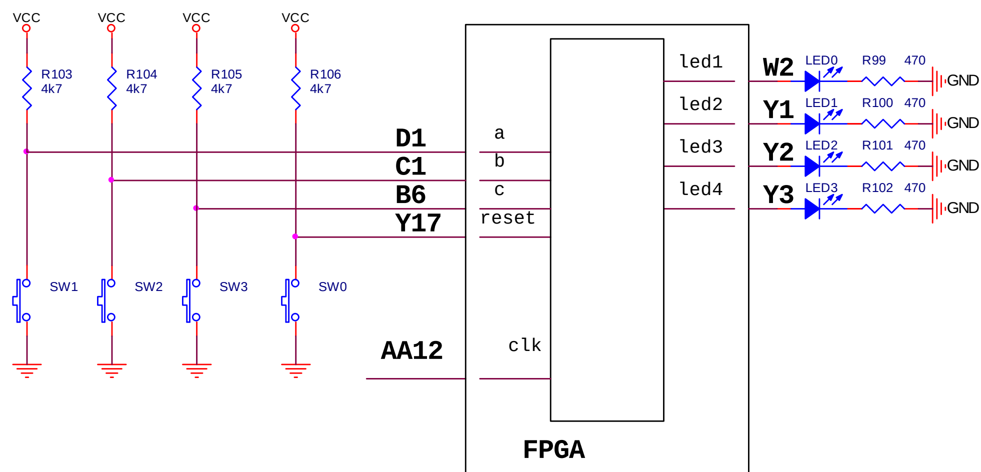
În ultimul proces se realizează decodarea stării următoare în funcție de portul de intrare. Semnalele de intrare **a**, **b** şi **c** sunt conectate într-un singur vector prin linia de cod:

```
data_in <= c & b & a;
```

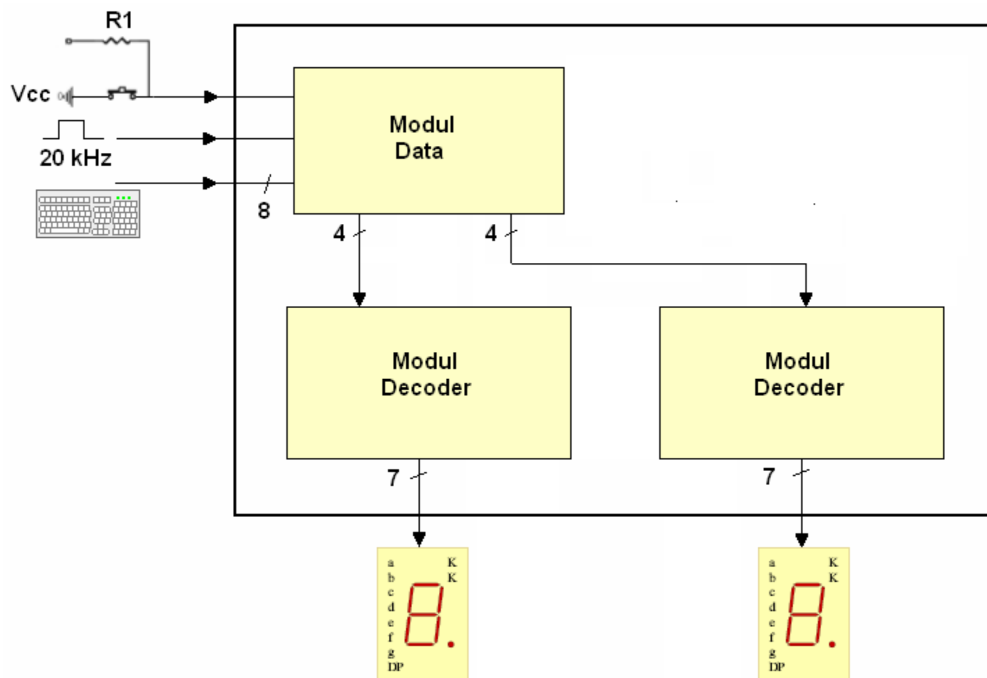
Este recomandat să se lucreze în acest mod pentru evitarea scrierii unor linii de cod prea mari.

Pentru verificarea funcționării acestui program este dată următoarea schemă electrică în care trebuie conectat modulul digital.

Semnalele **a**, **b** şi **c** sunt conectate la trei butoane SW0, SW1 şi SW2 active în '0' logic. Porturile de ieşire comandă patru leduri iar semnalul de ceas este preluat după sistemul de dezvoltare de la un oscilator de cuarț cu frecvența de 50MHz. Semnalul **reset** este conectat separat la butonul **test/reset**.



b) Să se implementeze prin cod VHDL un modul digital care preia datele de la o tastatură de tip ATX, le decodează şi le afişează pe două afişoare cu 7 segmente. Schema generală este dată în figura de mai jos:



Modulul digital ce urmează a fi realizat are două porturi de intrare şi două porturi de ieşire.

Pe porturile de intrare este conectat un buton de reset şi semnalele ce provin de la tastatură (**kbclk** şi **kbdata**) prin-un conector PS2. Porturile de ieşire comandă două afişoare cu leduri pe 7 segmente pe care sunt afişate valorile numerice ale tastelor apăsate în codul de scanare al tastaturii care este dat în figurile următoare:

Soluţie

Intern, modulul digital este implementat prin două module funcţionale independente şi interconectate printr-o descriere structurală:

- modul *Data* realizează preluarea datelor de tastatură serial în format binar după un semnal de ceas **clk**;
- modul *Decodor* realizează decodarea datelor binare în vederea afişării rezultatului pe doi digiţi ai sistemului reconfigurabil;

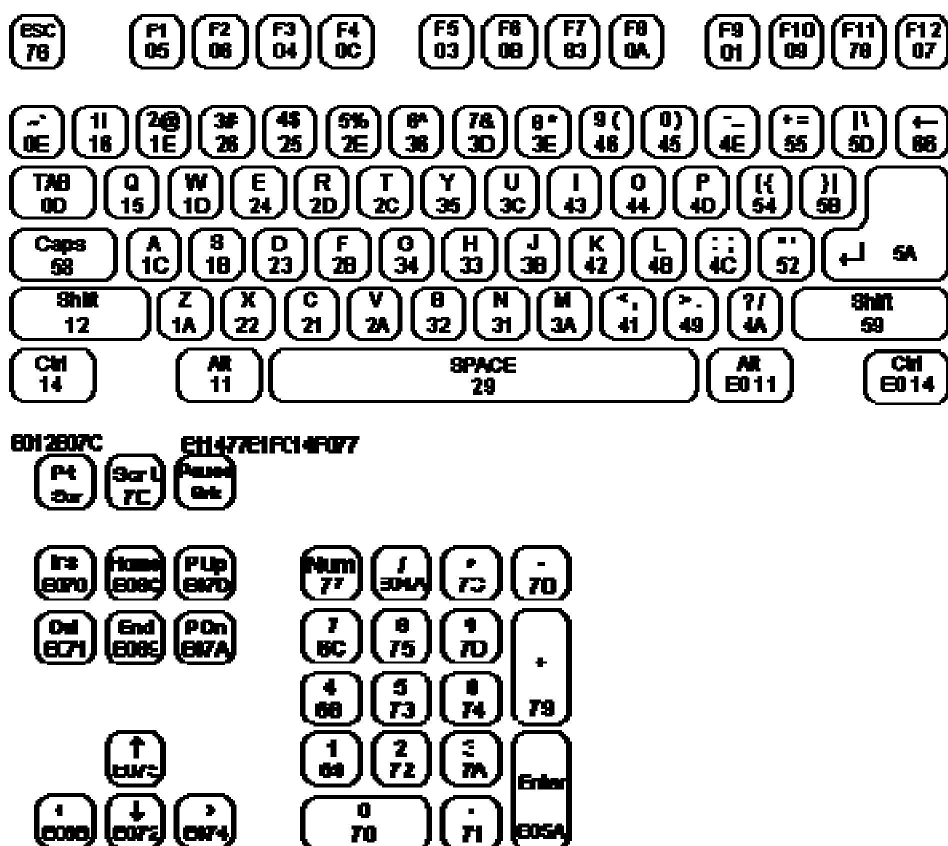
Tastatura

Tastatura este capabilă să trimită şi să recepţioneze comenzi, sa reţină date şi unele setări. Fiecărei taste îi este atribuit un cod scanabil. Acest cod, format din 2 cifre hexazecimale, este trimis de fiecare dată către unitatea care o gestionează atunci când o tastă este apăsată. Dacă o tasta este apăsată mai mult decât rata ei de tastare, atunci codul este trimis de mai multe ori pană când este încetată acţiunea asupra ei.

Un regim special îl au tastele "caps lock", "num lock" şi "scroll lock", ele având un led care semnalizează acţionarea tastei respective sau nu(asta în funcţie de starea ledului: aprins/stins).

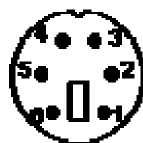
Codurile de scanare si conexiunea PS/2

În continuare este prezentă diagrama tastaturii şi codul de scanare pentru fiecare tastă, scris imediat sub fiecare tastă. Aceasta diagramă este preluată de la Beyond Logic Organization.

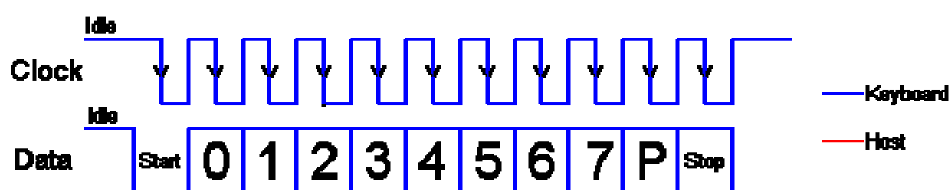


Pinii de conectare ai tastaturii la sunt următorii:

1. Clock
2. GND
3. Data
4. NC
5. VCC (+5V)
6. NC



De notat faptul că pe conectorul PS/2 se află un pin de ceas şi un pin de date. Datele binare se transmit serial sub forma următoare.



Generarea semnalului de ceas se realizează de către tastatură. Este în 1 logic atunci când nu se transmit date. La transmisia unei valori binare, datele sunt preluate pe frontul descrescător al acestuia.

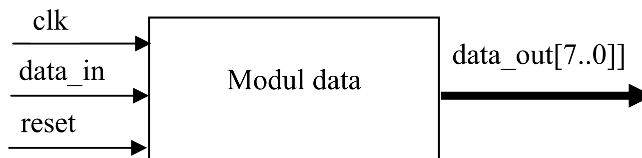
De notat faptul că datele sunt pe 8 biţi, transmise serial începând cu bitul cel mai puţin semnificativ şi în final este transmis un bit de paritate urmat de un bit de stop. Frecventa ceasului de la tastatură este în general între 20 şi 30 kHz.

Determinarea parităţii unui octet se realizează prin aflarea numărului de biţi cu valoarea 1 logic. Dacă acesta este un număr par, atunci octetul este par iar dacă numărul de biţi în 1 logic este impar, octetul respectiv este impar. O determinare rapidă a parităţii unui octet se realizează prin aplicarea operatorului XOR între toţi biţii octetului.

$$paritate = bit_7 \oplus bit_6 \oplus bit_5 \oplus bit_4 \oplus bit_3 \oplus bit_2 \oplus bit_1 \, XOR \oplus bit_0$$

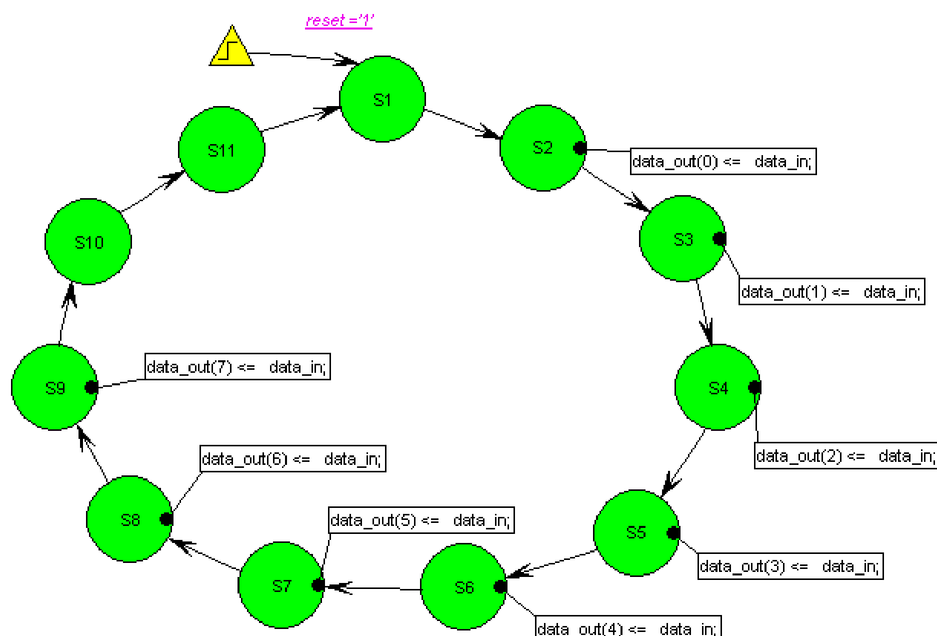
Ultimul bit **P** din şirul transmis de către tastatură este utilizat în transmiterea unui cuvânt numai cu o anumită paritate. De exemplu, dacă se doreşte transmiterea pe paritate pară, în cazul în care octetul este par, bitul de paritate este 0 logic, dacă octetul este impar, bitul de paritate este 1 logic şi cuvântul rezultat (prin efectuarea operaţiei de XOR între toţi biţii ai octetului inclusiv al bitului **P** rezultă un cuvânt par).

Modulul Data permite obţinerea codului binar a tastei apăsate şi prezintă următoarele porturi:



- **reset**: este pe 1 bit, folosit pentru inițializarea automatului cu stări finite în cazul în care sistemul nu funcționează corect;
- **data_in**: semnal pe 1 bit ce reprezentă datele preluate de la tastatură în ritmul semnalului de ceas generat de tastatură;
- **clk** : semnal de ceas provenit de la tastatură pentru sincronizarea datelor transmise pe semnalul **data_in**. Datele sunt preluate pe frontul negativ al ceasului.

Acest modul este implementat cu ajutorul unui automat cu stări finite care respectă următorul graf de stări:



Se observă că în acest graf de stări nu se ține cont de bitul de paritate. Acest modul va fi îmbunătățit după îndrumările date în ultimul capitol al acestui laborator.

Codul sursă în limbajul VHDL este următorul:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity data is
   Port ( reset : in std_logic;
                              clk : in std_logic;
```

```
        data_in : in std_logic;
        data_out : out std_logic_vector(7 downto 0));
end data;

architecture data_arch of data is
        type tip_stare is (st1, st2, st3, st4, st5, st6, st7, st8, st9, st10, st11);
        signal stare, stare_urm: tip_stare;
begin
        process(clk, reset)
        begin
                if reset ='0' then
                        stare <= st1;
                elsif (clk'event and clk='0')then
                        stare <= stare_urm;
                end if;
        end process;
        process(stare, data_in)
        begin
                case(stare) is
                        when st2 =>     data_out(0) <=  data_in;
                        when st3 =>     data_out(1) <=  data_in;
                        when st4 =>     data_out(2) <=  data_in;
                        when st5 =>     data_out(3) <=  data_in;
                        when st6 =>     data_out(4) <=  data_in;
                        when st7 =>     data_out(5) <=  data_in;
                        when st8 =>     data_out(6) <=  data_in;
                        when st9 =>     data_out(7) <=  data_in;
                        when others => null;
                end case;
        end process;
        process(stare, data_in)
        begin
                case(stare) is
                        when st1 =>     stare_urm <= st2;
                        when st2 =>     stare_urm <= st3;
                        when st3 =>     stare_urm <= st4;
                        when st4 =>     stare_urm <= st5;
                        when st5 =>     stare_urm <= st6;
                        when st6 =>     stare_urm <= st7;
                        when st7 =>     stare_urm <= st8;
                        when st8 =>     stare_urm <= st9;
                        when st9 =>     stare_urm <= st10;
                        when st10 => stare_urm <= st11;
                        when st11 => stare_urm <= st1;
                        when others => null;
                end case;
        end process;
end data_arch;
```

Acest program este compus din trei procese. În primul proces este actualizată starea prezentă în funcție de starea următoare după cum s-a arătat în partea teoretică a acestui laborator.

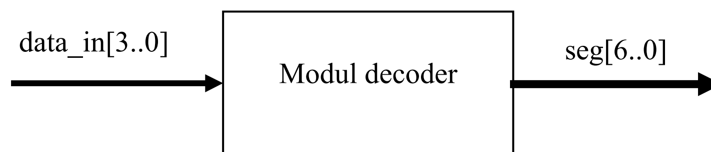
Cel de-al doilea proces este utilizat doar pentru decodarea portului de ieşire în funcție de stări. Practic, prin el se realizează un circuit combinațional senzitiv la semnalul **stare** şi portul de intrare **data_in**. Decodarea a fost realizată prin intermediul unei specificații secvențiale de tip **case**.

În cazul în care sunt realizate automate de tip Mealy, specificația CASE poate fi înlocuită cu o secvență de specificații de tip IF.

Al treilea proces realizează decodarea stărilor automatului secvențial. Acest lucru se face tot printr-o specificație de tip CASE.

În partea teoretică a fost prezentat un automat cu stări finite format din două procese. În acest exemplu este prezentat un automat cu trei procese pentru a arăta faptul că acest sistem este mult mai bine optimizat prin această metodă şi nu există un standard de realizare a automatelor cu stări finite.

Modulul decoder este implementat pentru a realiza transformarea din forma numerică binară într-un cod pe 7 biți care să reprezinte cifrele hexazecimale de afişoarele cu 7 segmente. Acest modul are următoarea configurație:



Codul sursă VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

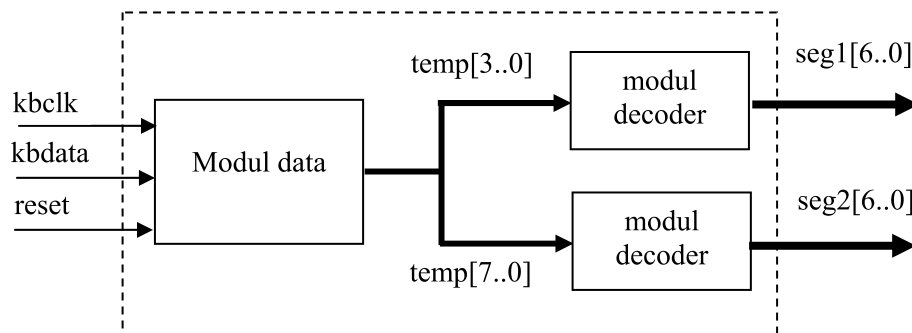
entity decoder is
   Port ( data_in : in std_logic_vector(3 downto 0);
          seg : out std_logic_vector(6 downto 0));
end decoder;

architecture Behavioral of decoder is
begin
        process(data_in)
        begin
                CASE (data_in) IS
                        WHEN "0000"=> seg <="0111111";
                        WHEN "0001"=> seg <="0000110";
```

```
                    WHEN "0010"=> seg <="1011011";
                    WHEN "0011"=> seg <="1001111";
                    WHEN "0100"=> seg <="1100110";
                    WHEN "0101"=> seg <="1101101";
                    WHEN "0110"=> seg <="1111101";
                    WHEN "0111"=> seg <="0000111";
                    WHEN "1000"=> seg <="1111111";
                    WHEN "1001"=> seg <="1101111";
                    WHEN "1010"=> seg <="1110111";
                    WHEN "1011"=> seg <="1111100";
                    WHEN "1100"=> seg <="0111001";
                    WHEN "1101"=> seg <="1011110";
                    WHEN "1110"=> seg <="1111001";
                    WHEN "1111"=> seg <="1110001";
                    WHEN OTHERS=> null;
               END CASE;
          end process;
     end Behavioral;
```

Decodarea este realizată printru proces care implementează un circuit combinațional utilizând specificația secvențială de tip CASE. Acest modul poate fi realizat integral cu specificația concurentă selectivă WITH/SELECT.

Integrarea celor două module într-un singur circuit se realizează printr-o descriere structurală după schema bloc din figura de mai jos:



Codul sursă VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity main_tastatura is
  Port ( reset : in std_logic;
                           kbclk : in std_logic;
       kbdata : in std_logic;
```

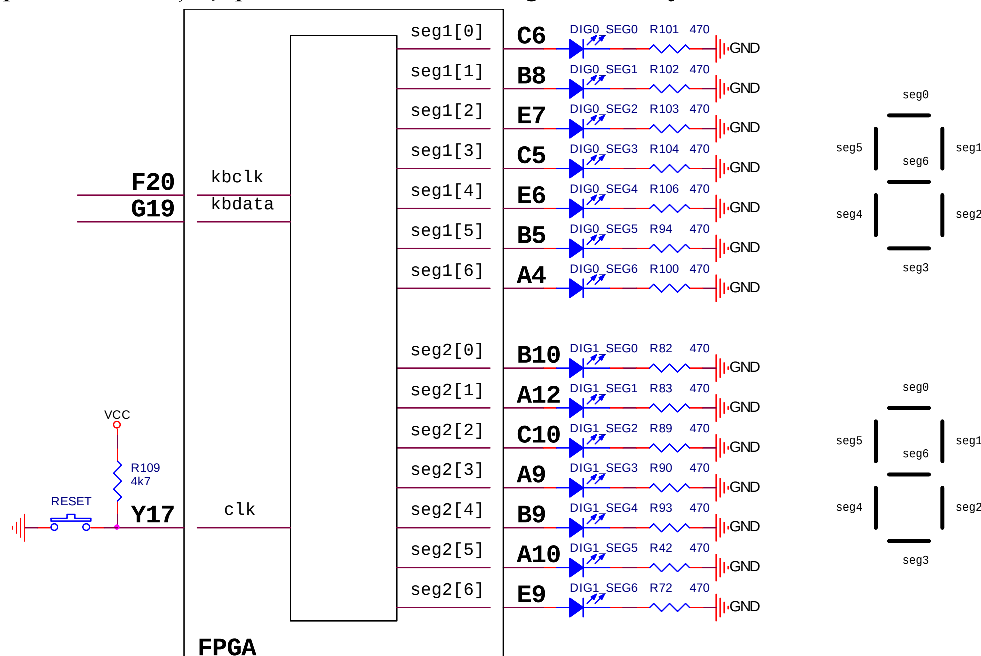
```
        seg1 : out std_logic_vector(6 downto 0);
        seg2 : out std_logic_vector(6 downto 0));
end main_tastatura;

architecture Behavioral of main_tastatura is
    component decoder is
    Port (  data_in : in std_logic_vector(3 downto 0);
        seg : out std_logic_vector(6 downto 0));
    end component;

    component data is
    Port ( reset : in std_logic;
            clk : in std_logic;
            data_in : in std_logic;
            data_out : out std_logic_vector(7 downto 0));
    end component;

    signal temp: std_logic_vector(7 downto 0);
begin
    U1: decoder port map(temp(7 downto 4), seg2);
    U2: decoder port map(temp(3 downto 0), seg1);
    U3: data port map(reset,kbclk, kbdata, temp);
end Behavioral;
```

Schema electrică pe care este implementat acest circuit şi legătura între porturile entităţii şi pinii fizici sunt date în figura de mai jos:



Modulul data poate fi realizat şi printr-o descriere comportamentală ca în programul VHDL de mai jos:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity data is
    Port ( reset : in std_logic;
           clk : in std_logic;
          data_in : in std_logic;
          data_out : out std_logic_vector(7 downto 0));
end data;

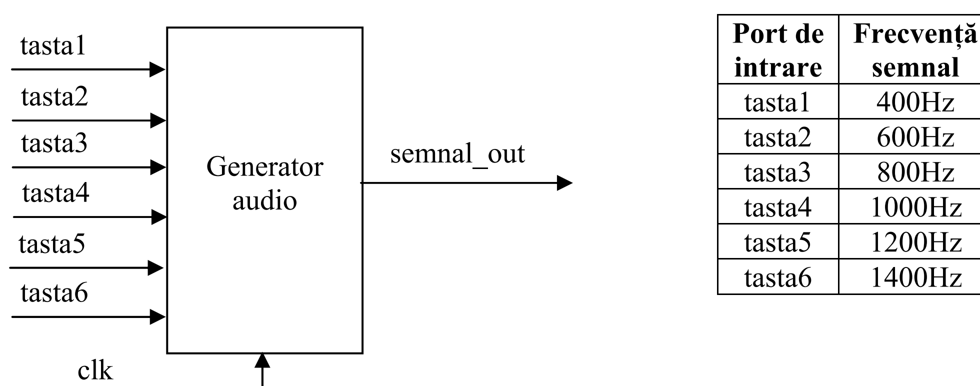
architecture data_arch of data is
begin
        process (clk, reset)
                variable contor: natural range 0 to 10;
                variable temp: std_logic_vector(9 downto 0);
        begin
                if reset='0' then contor:=0;
                elsif falling_edge(clk) then
                        if contor=10 then
                                contor:=0;
                                data_out <= temp(8 downto 1);
                        else
                                contor:=contor+1;
                                temp:=data_in & temp(9 downto 1);
                        end if;
                end if;
        end process;
end data_arch;
```

Descrierea comportamentală a acestui modul se realizează prin intermediul unui proces. Acest proces conţine în lista de senzitivităţi două semnale **clock** şi **reset** ce permit activarea procesului. În cadrul procesului sunt declarate două variabile: *numarator* ce permite numărarea clock-ului generat de tastatura şi *index* folosită pentru setarea poziţiei în vectorul de date a biţilor recepţionaţi de la tastatură. Cazurile de excepţie ale variabilei numărător sunt:

- când are valoarea 10: tastatura a terminat de transmis întregul cod al tastei iar variabila este resetată pentru o numărare ulterioară;
- când are valorile 0 şi 9: înseamnă că sunt transmişi biţii de start, respectiv stop iar variabila trebuie incrementată pentru obţinerea biţilor de date;

- când are alte valori decât cele menționate atunci când biții transmişi sunt cei de date.

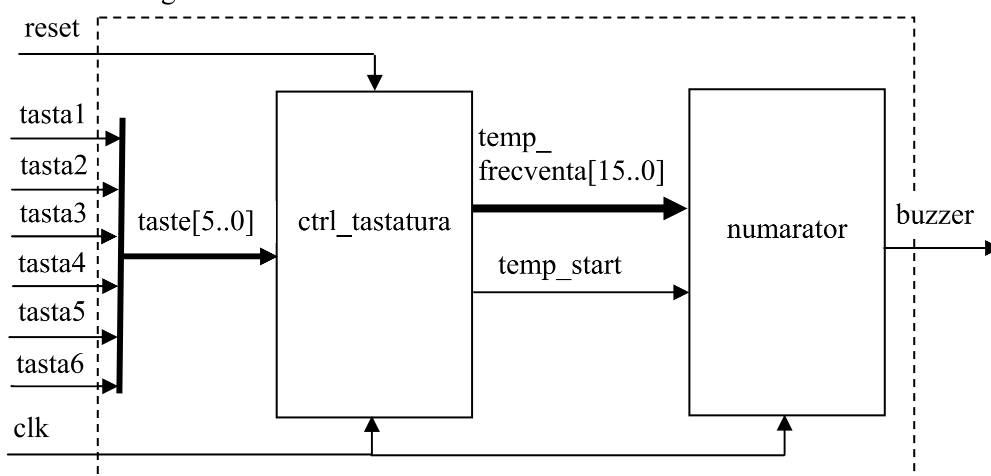
c) Să se descrie în limbajul VHDL un generator audio care poate emite 6 semnale cu frecvențe diferite. Generatorul audio prezintă 6 semnale de intrare pe câte un singur bit şi sunt active în 0 logic. Activarea unuia din semnalele de intrare conduce la emiterea unui sunet audio cu o anumită frecvență. Frecvența de ceas a sistemului este de 50MHz. Porturile modulului digital şi tabelul de asociere dintre porturile de intrare şi frecvențele semnalelor audio sunt reprezentate prin figura de mai jos:



Port de intrare	Frecvență semnal
tasta1	400Hz
tasta2	600Hz
tasta3	800Hz
tasta4	1000Hz
tasta5	1200Hz
tasta6	1400Hz

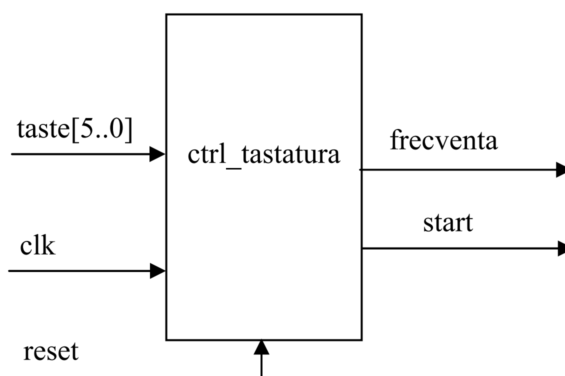
Soluție

Generatorul audio a fost realizat din două module. Primul modul are funcția de stabilire a frecvențelor semnalelor audio ce vor fi generate în funcție de tasta apăsată iar al doilea modul generează frecvența dorită. Schema bloc structurală a generatorului audio este următoarea:



Modulul ctrl_tastatura

Acest modul este realizat printr-un automat cu stări finite. Are ca semnale de intrare: semnalul **reset**, asincron, activ în 0 logic, prin care se aduce automatul în starea inițială, magistrala **taste** după care se iau deciziile în automat şi semnalul **clk**. Pe semnalele de ieşire sunt transmise: constanta de divizare (frecventa ce trebuie generată) numărătorului şi semnalul start prin care este activat sau dezactivat numărătorul.

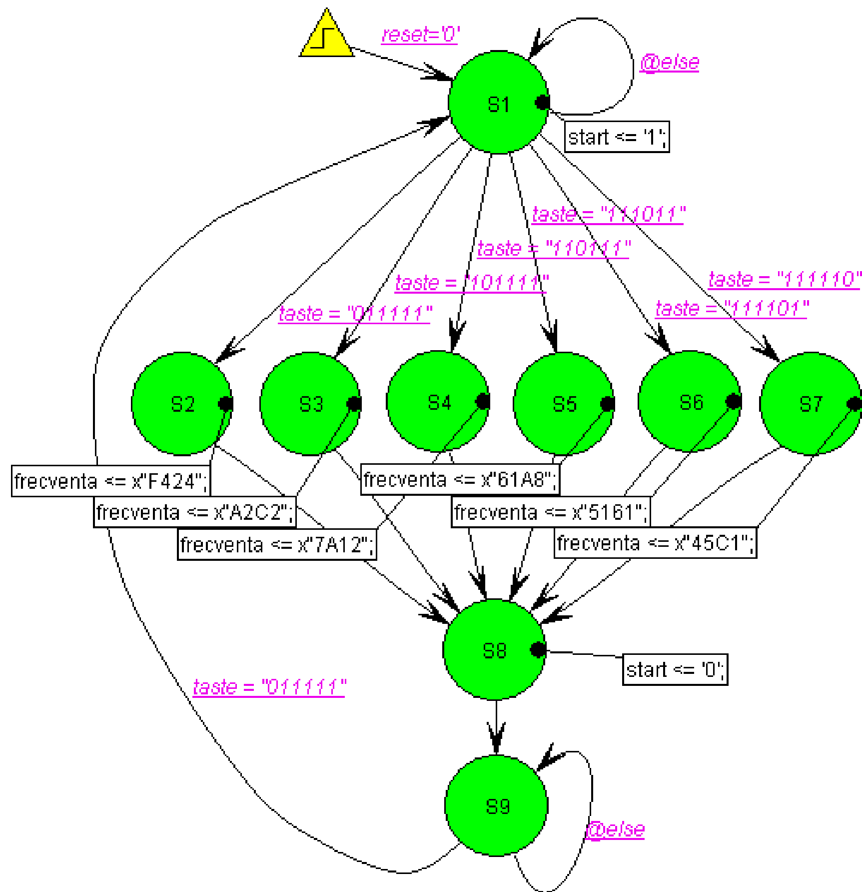


Graful de stări care caracterizează modulul **ctrl_tastatura** este dat în figura din pagina următoare.

Dacă nu este nici-o tastă apăsată, automatul rămâne în starea **st1**. Ieşirea **start** este în 1 logic ceea ce înseamnă că numărătorul va fi inactiv. Intrarea în una din stările **st2** .. **st7** se face numai la apăsarea uneia din cele 6 taste. În stările **st2** .. **st7** se atribuie constanta de divizare semnalului frecvență după care, în starea **st8** se activează numărătorul prin trecerea semnalului **start** în 0 logic. Se trece apoi în starea **st9** şi automatul va fi blocat în această stare până când nu mai este apăsată nici-o tastă după care se reia secvența cu starea **st1**.

Constantele de divizare sunt calculate prin împărțirea frecvenței sistemului, (50MHz) la de două ori mai mare frecvența dorită după care constanta zecimală obținută este trecută în hexazecimal.

Frecvență semnal	**Constantă de divizare**
400Hz	F424h
600Hz	A2C2h
800Hz	7A12h
1000Hz	61A8h
1200Hz	5161h
1400Hz	45C1h



Descrierea automatului cu stări finite este descris în limbajul VHDL prin codul prezentat mai jos:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

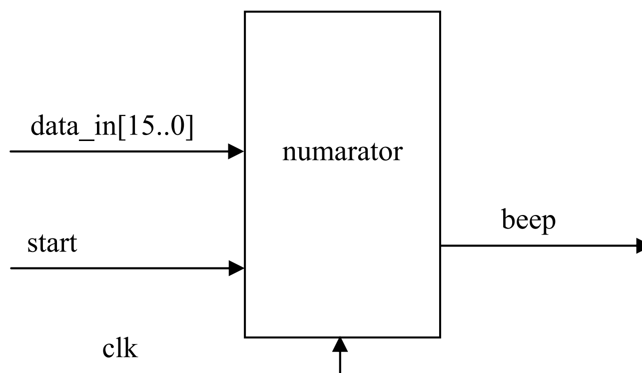
entity ctrl_tastatura is
   Port ( clk : in std_logic;
          reset : in std_logic;
                                 taste : in std_logic_vector(5 downto 0);
                                 start  : out std_logic;
          frecventa : out std_logic_vector(15 downto 0));
end ctrl_tastatura;
```

```
architecture Behavioral of ctrl_tastatura is
        type tip_stare is (st1, st2, st3, st4, st5, st6, st7, st8, st9);
        signal stare: tip_stare;
begin


        process(clk, reset)
        begin
        if (reset='0') then
                stare <= st1;
        elsif (clk'event and clk='1') then
                case (stare) is
                        when st1 =>
                                start <= '1';
                                if    (taste = "011111")  then stare <= st2;
                                elsif  (taste = "101111") then stare <= st3;
                                elsif  (taste = "110111") then stare <= st4;
                                elsif  (taste = "111011") then stare <= st5;
                                elsif  (taste = "111101") then stare <= st6;
                                elsif  (taste = "111110") then stare <= st7;
                                else  stare <= st1;
                                end if;
                        when st2 =>     stare <= st8;
                                        frecventa <= x"F424";
                        when st3 =>     stare <= st8;
                                        frecventa <= x"A2C2";
                        when st4 =>     stare <= st8;
                                        frecventa <= x"7A12";
                        when st5 =>     stare <= st8;
                                        frecventa <= x"61A8";
                        when st6 =>     stare <= st8;
                                        frecventa <= x"5161";
                        when st7 =>     stare <= st8;
                                        frecventa <= x"45C1";
                        when st8 =>     stare <= st9;
                                        start <= '0';
                        when st9 =>
                                if taste="111111" then
                                        stare <= st1;
                                else
                                        stare <= st9;
                                end if;
                        when others => null;
                end case;
        end if;
        end process;
end Behavioral;
```

Modulul numarator

Acest modul este un numărător clasic activat prin semnalul **start** şi numără până la valoarea dată pe portul de intrare **data_in**. La terminarea unei numărări complementează valoarea semnalului de ieşire **beep**.



Programul sursă VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity numarator is
   Port (          clk: in std_logic;
                                  data_in : in std_logic_vector(15 downto 0);
          start   : in std_logic;
          beep    : out std_logic);
end numarator;

architecture Behavioral of numarator is
         signal iesire  : std_logic;

begin

         process (clk,start, data_in)
                  variable contor  : std_logic_vector(15 downto 0);
         begin
                  if (clk'event and clk='1') then
                           if (start='1') then
                                    contor := (others => '0');
                                    iesire <= '0';
                           elsif (contor = x"0000") then
```

```
                    contor := data_in;
                    iesire <= not iesire;
                else
                    contor := contor - '1';
                end if;
            end if;
        end process;

        beep <= iesire;

    end Behavioral;
```

Semnalul **start** este sincron cu semnalul de ceas. Numărarea se realizează prin decrementarea variabilei **contor** şi reiniţializarea acesteia cu valoarea semnalului **data_in** la trecerea prin zero.

Cele două module **ctrl_tastatură** şi **numărător** sunt interconectate printr-o descriere de tip structurală dată în următorul program VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity generator_sunet is
   Port ( clk : in std_logic;
                    reset: in std_logic;
        tasta1 : in std_logic;
        tasta2 : in std_logic;
        tasta3 : in std_logic;
        tasta4 : in std_logic;
        tasta5 : in std_logic;
        tasta6 : in std_logic;
        difuzor : out std_logic);
end generator_sunet;

architecture Behavioral of generator_sunet is
         component ctrl_tastatura is
   Port ( clk : in std_logic;
        reset : in std_logic;
                    taste : in std_logic_vector(5 downto 0);
                    start  : out std_logic;
        frecventa : out std_logic_vector(15 downto 0));
         end component;

         component numarator is
```

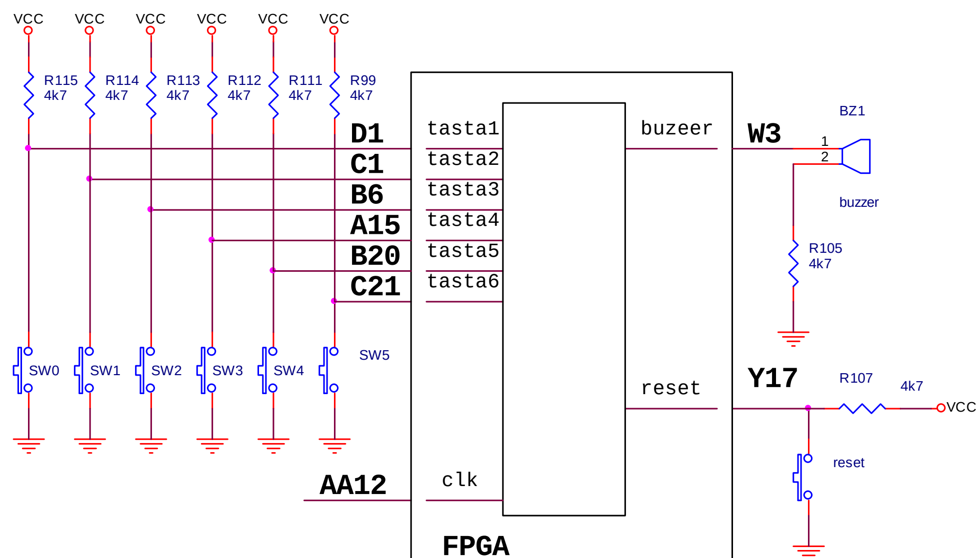
```
Port (          clk: in std_logic;
                                          data_in : in std_logic_vector(15 downto 0);
        start : in std_logic;
        beep : out std_logic);
         end component;

         signal temp_frecventa: std_logic_vector(15 downto 0);
         signal temp_start: std_logic;
         signal temp_taste: std_logic_vector(5 downto 0);

begin
         temp_taste <= tasta1 & tasta2 & tasta3 & tasta4 & tasta5 & tasta6;
         U1: ctrl_tastatura port map(clk, reset, temp_taste, temp_start,
temp_frecventa);
         U2: numarator port map(clk, temp_frecventa, temp_start, difuzor);

end Behavioral;
```

Schema electrică minimală în care va fi încadrat generatorul audio realizat este dată în figura de mai jos.



3. Exerciții

1. În vederea implementării programelor de mai sus să se realizeze următorii paşi:
   - Pentru fiecare aplicație în parte să se realizeze un proiect nou;
   - Introducerea surselor VHDL;
   - Simularea logică a proiectelor;
   - Crearea fişierelor de constrângeri ale pinilor după schemele de implementare fizică;
   - Realizarea sintezei acestora;
   - Vizualizarea schemelor logice, respectiv tehnologice;
   - Simularea modelului comportamental rezultat după sinteză;
   - Implementarea proiectului;
   - Vizualizarea rutării structurii fizice;
   - Simularea Post-Place & Post Route a modelului;
   - Configurarea circuitului fizic.

2. La cea de a doua aplicație (decodorul de tastatură) să se realizeze următoarele:

   a) faceți o comparație între cele două implementări: comportamentale, respectiv cu FSM;

   b) Adăugați la program corecția parității datelor trimise de către tastatură;

   c) Afişați codul ASCII ale caracterelor corespunzătoare şi nu codul tastei respective prin realizarea modulului de conversie din codul tastaturii în codul ASCII doar pentru caracterele alfanumerice (A .. Z, 0 .. 9).

TABELA DE CODURI ASCII
(AMERICAN STANDARD CODE FOR INTERCHANGE INFORMAȚION)

---	0	1	2	3	4	5	6	7
0	NUL	DLE	SPACE	0	@	P	/	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	‘	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O		o—	DEL

3. În aplicația referitoare la generatorul de sunet să se aducă următoarele modificări:

a) Pentru fiecare buton în parte SW0 .. SW5 să i se atribuie câte un led LED0 .. LED5. La apăsarea butonului, pe lângă sunetul generat să se aprindă şi ledul corespunzător;

b) Modificați frecvențele astfel încât la apăsarea succesivă a butoanelor să se genereze notele unei octave

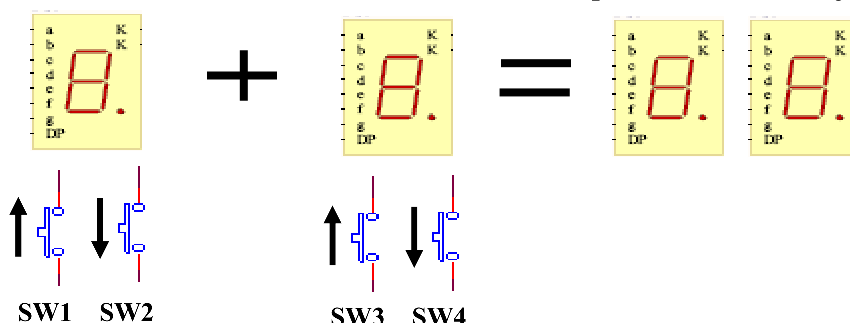
Octava	Do	Re	Mi	Fa	Sol	La	SI
1	18357	16345	14551	13715	12175	10847	9701
2	9108	8117	7231	6818	6088	5424	4831
3	4554	4058	3616	3409	3044	2712	2415
4	2277	2096	1808	1705	1522	1356	1208
5	1139	1015	904	852	761	678	604
6	569	507	452	426	380	339	302
7	285	254	226	213	190	169	151

În cadrul problemei noastre vom considera notele cuprinse în octava a doua. Pentru această octavă valorile de inițializare, în hexazecimal, ale generatorului de timp sunt următoarele:

Do	Re	Mi	Fa	Sol	La	SI	Do
2394	1FB5	1C3F	1AA2	17C8	1530	12DF	11CA

4. Realizați un modul digital descris în VHDL care să emită o secvență de 10 note muzicale generate la interval de 1 secundă.

5. Să se implementeze un sumator a doi operanzi reprezentați pe câte un digit. Introducerea numerelor se face prin operația de incrementare, respectiv decrementare a valorilor numerice afişate pe digiți la apăsarea tastelor SW1..SW4. Rezultatul este afişat instant pe următorii doi digiți.



Asocierea dintre pinii structurii FPGA şi porturile entității modulului digital se face după schema electrică a sistemului digital.

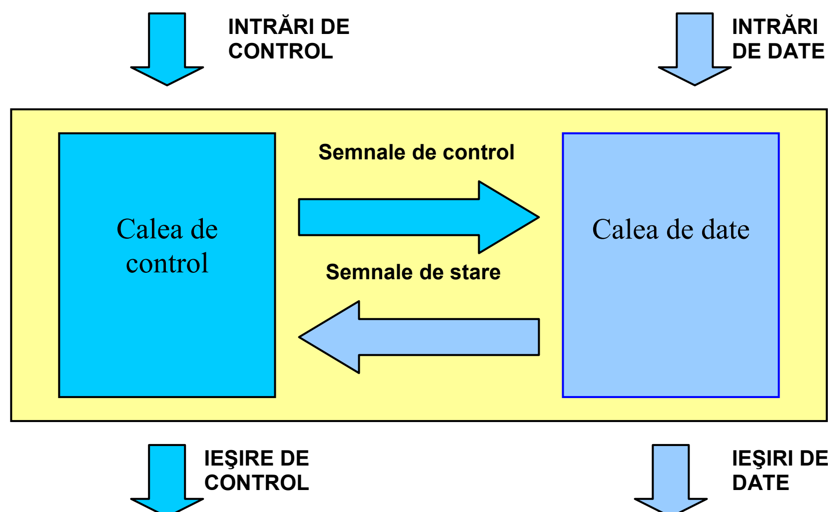
Capitolul 6
Model de proiectare a unui modul digital în limbajul de descriere hardware VHDL

În acest capitol este prezentată proiectarea unui sistem digital. Ca şi aplicaţie este descris un modul digital alcătuit dintr-un ansamblu de (sub)componente interconectate ce conlucrează în scopul realizării activităţii de control asupra unui proces. În vederea proiectării cât mai eficiente şi rapide a acestui modul, subcomponentele au fost descrise prin diferite tehnici de implementare (prin cod VHDL, diagrame cu stări finite sau schematic la nivel de blocuri).

1. Breviar teoretic

Proiectarea unui sistem digital constă, în primul rând, în formularea specificaţilor sistemului digital şi descrierea funcţionalităţii acestuia. Urmează faza prin care se realizează proiectarea efectivă prin descrieri HDL susţinută de simulări pentru verificarea corespondenţei dintre rezultatele obţinute şi cerinţele sistemului. Faza finală este aceea de implementare a sistemului digital într-o structură digitală programabilă. Este recomandat ca proiectarea propriu-zisă a unui sistem digital de complexitate mărită să fie realizată pe module.

Modulele pot fi structurate în două categorii:
- module utilizate în controlul sistemului digital (calea de control);
- module utilizate în scopul asigurării funcţiei de transfer între porturile de date I/O (calea de date).



Modulele digitale de temporizatoare sau numărare vor fi incluse în calea de date şi vor fi comandate cu semnale provenite de la modulele din calea de control.

Această modalitate de proiectare conduce la posibilitatea personalizării în moduri diferite a căii de date, respectiv a căii de control. Facilitează verificarea proiectului prin testarea căii de control separat de calea de date.

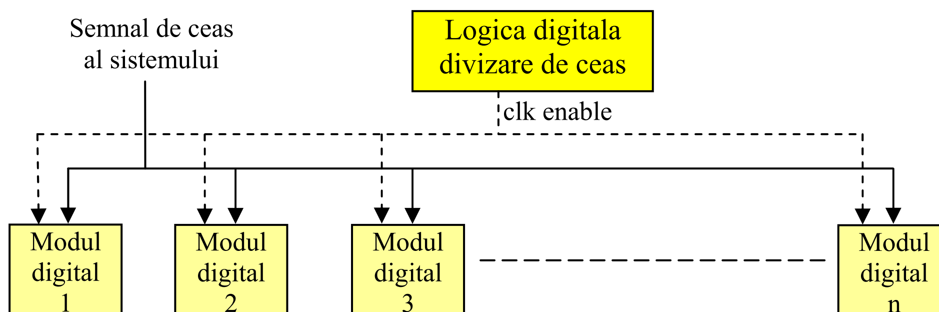
Este recomandat ca **implementarea căii de date** să fie realizată prin specificaţii concurente sau procese combinaţionale. Datele rezultate din acestea vor fi salvate în regiştrii. Calea de date se va realiza structural pe o arhitectură bine modularizată pe mai multe componente, eventual componentele pe mai multe submodule sau procese.

Calea de control poate fi modelată prin automate de stări finite de tip Mealy sau Moore, respectiv cu întârziere sau imediat. Se recomandă modelarea unui automat Moore imediat pentru sisteme în care timpul nu reprezintă o constrângere sau Mealy cu întârziere în cazuri când este necesar un număr mai mic de stări.

În vederea realizării unui proiect complex se recomandă respectarea următorilor paşi de lucru:

- determinarea funcţiei realizate de către calea de date;
- stabilirea semnalelor de intrare/ieşire pentru date şi a semnalelor de control;
- realizarea organigramelor pentru descrierea comportamentală (se urmăreşte o implementare cât mai directă a acestora prin automate cu stări finite) a căii de control;
- stabilirea semnalelor de interfaţare a căii de control cu exteriorul;
- descrierea prin cod VHDL şi simularea căii de date şi control;
- implementarea în structura reconfigurabilă.
-

Foarte important, în proiectarea sistemelor digitale, este asigurarea corectă a semnalului de ceas pentru fiecare modul în parte. Este obligatoriu, pentru evitarea hazardului logic, ca să nu se interpună pe semnalul de ceas structuri de porţi logice şi să se ajungă la eroarea de tip „gated clock" dată de către sintetizator .



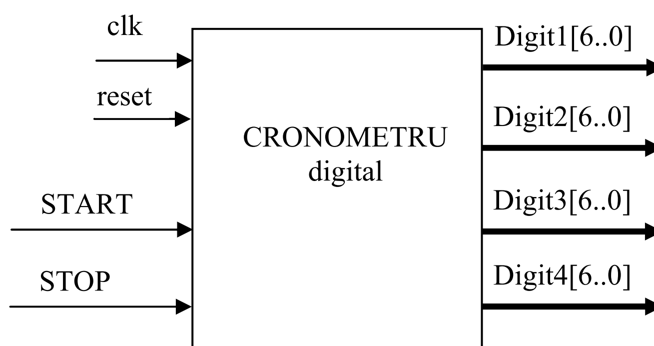
După cum este arătat in figura de mai sus, semnalul de ceas a sistemului se aplică fiecarui modul in parte iar pentru activarea acestuia se introduce un semnal de tip *clk_enable*.

2. Aplicație

Să se realizeze prin descriere în cod VHDL un cronometru digital pe sistemul de dezvoltare din laborator care are următoarele specificații:

- afişarea timpului să se realizeze pe patru digiți (zeci de secunde, secunde, zecimi de secundă, sutimi de secundă);
- să prezinte trei taste de comandă (START, STOP şi Reset)

Entitatea modulul electronic cu porturile I/O este prezentată în figura de mai jos:



Porturile de intrare START şi STOP sunt utilizate pentru pornirea, respectiv oprirea temporizării. Dacă se doreşte o nouă temporizare este necesară resetarea modulului digital.

Porturile de ieşire numite **digit1 … digit6** vor fi conectate la patru afişoare cu 7 segmente. Apăsarea butonului **reset** conduce la afişarea valorii numerice 0000.

Semnalul **clk** este preluat de la oscilatorul de cuarț disponibil în sistemul de dezvoltare şi are frecvența de 50MHz.

Schema electrică în care se va încadra modulul digital este dată în figura din pagina următoare.

VCC
R66
4k7
Y17
reset
SW1

AA12
clk
50MHz

VCC
R95
4k7
D1
START
SW5

VCC
R96
4k7
C1
STOP
SW6

seg1[0] C6 DIG0_SEG0 R67 470 GND
seg1[1] B8 DIG0_SEG1 R68 470 GND
seg1[2] E7 DIG0_SEG2 R69 470 GND
seg1[3] C5 DIG0_SEG3 R70 470 GND
seg1[4] E6 DIG0_SEG4 R71 470 GND
seg1[5] B5 DIG0_SEG5 R64 470 GND
seg1[6] A4 DIG0_SEG6 R65 470 GND

seg2[0] B10 DIG1_SEG0 R59 470 GND
seg2[1] A12 DIG1_SEG1 R60 470 GND
seg2[2] C10 DIG1_SEG2 R61 470 GND
seg2[3] A9 DIG1_SEG3 R62 470 GND
seg2[4] B9 DIG1_SEG4 R63 470 GND
seg2[5] A10 DIG1_SEG5 R34 470 GND
seg2[6] E9 DIG1_SEG6 R58 470 GND

seg3[0] E15 DIG2_SEG0 R90 470 GND
seg3[1] E16 DIG2_SEG1 R91 470 GND
seg3[2] A14 DIG2_SEG2 R92 470 GND
seg3[3] D14 DIG2_SEG3 R93 470 GND
seg3[4] D13 DIG2_SEG4 R94 470 GND
seg3[5] B13 DIG2_SEG5 R88 470 GND
seg3[6] E13 DIG2_SEG6 R89 470 GND

seg4[0] B17 DIG3_SEG0 R83 470 GND
seg4[1] B18 DIG3_SEG1 R84 470 GND
seg4[2] A18 DIG3_SEG2 R85 470 GND
seg4[3] B15 DIG3_SEG3 R86 470 GND
seg4[4] D17 DIG3_SEG4 R87 470 GND
seg4[5] C17 DIG3_SEG5 R72 470 GND
seg4[6] E17 DIG3_SEG6 R82 470 GND

seg0
seg5 seg6 seg1
seg4 seg2
seg3

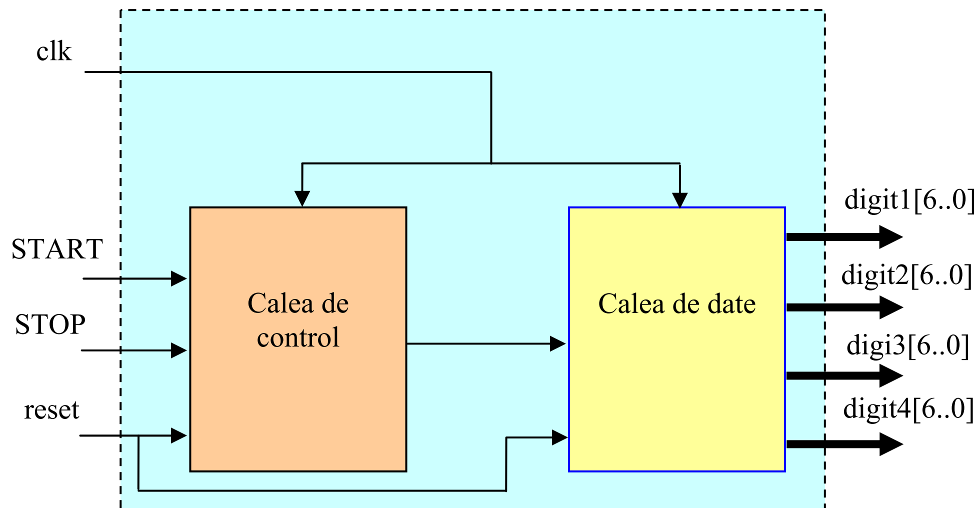
seg0
seg5 seg6 seg1
seg4 seg2
seg3

seg0
seg5 seg6 seg1
seg4 seg2
seg3

seg0
seg5 seg6 seg1
seg4 seg2
seg3

FPGA

Implementarea acestui proiect se face modular fiind divizat în două părți: calea de control şi calea de date.



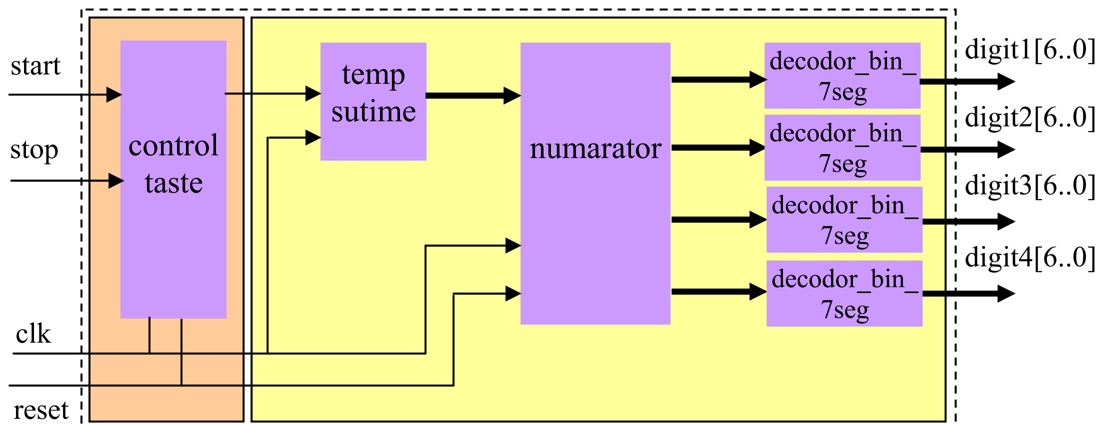
Porturile de intrare ale căii de date provin de la calea de control. Are doar semnale de ieşire pentru date, fiind liniile de aprindere ale afişajelor cu şapte segmente.

Calea de control are semnale de intrare porturile **START**, **STOP** şi **reset**.

Distribuția semnalului de ceas se face în formă de arbore pentru evitarea hazardului logic.

Implementarea proiectului

Prezentarea descrierii proiectului se face după o topologie de tip TOP-DOWN. Schema de descriere structurală a cronometrului este următoarea:



Calea de control este formată din modulul:

- **control_taste**.

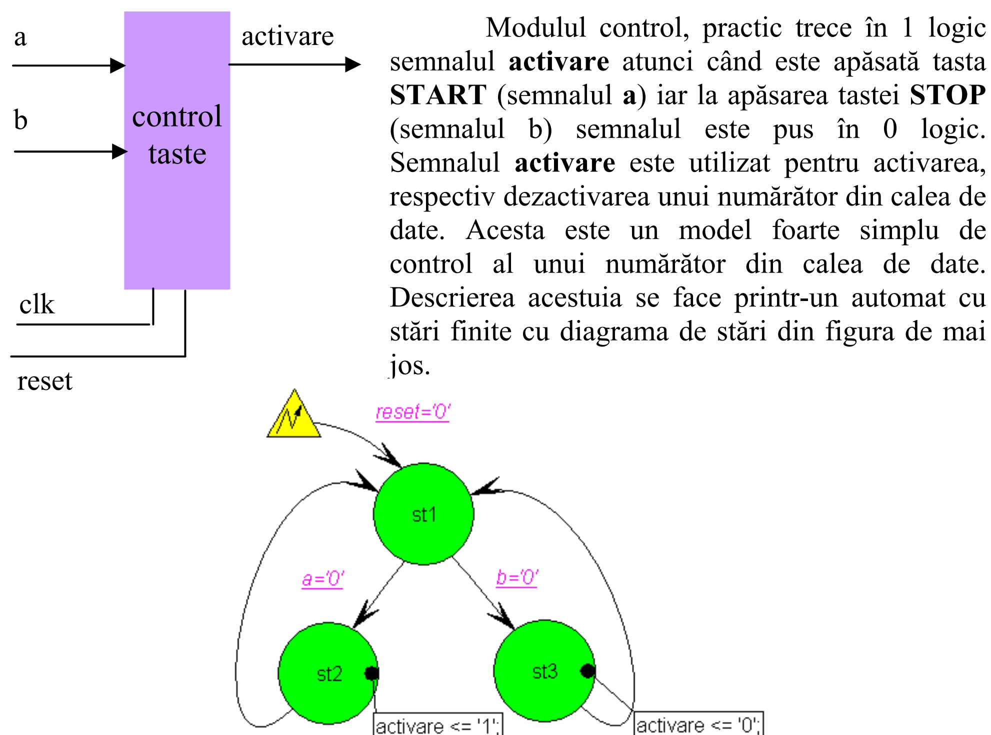
Calea de date este formată din modulele:

- **temp_sutime**;
- **numărător**;
- **decodor_bin_7seg**.

Prin calea de control se activează sau dezactivează funcționarea modului **temp_sutime** din calea de date în funcție de combinația tastelor de intrare. Dacă este activat semnalul **start**, atunci calea de date funcționează, în caz contrar, când semnalul **stop** este activ, aceasta păstrează pe semnalele de ieşire ultima valoare rămasă. În calea de date sunt utilizate trei componente, prima pentru obținerea sutimii de secundă, a doua pentru numărarea până la 60 de secunde pe 4 digiți iar ultima componentă este pentru a decoda valorile binare în forma vizuală a cifrelor corespondente pe afişoare cu 7 segmente.

Modulul control_taste

Porturi de intrare/ieşire ale modulul de control pentru taste este dat în figura de mai jos:



Modulul control, practic trece în 1 logic semnalul **activare** atunci când este apăsată tasta **START** (semnalul **a**) iar la apăsarea tastei **STOP** (semnalul b) semnalul este pus în 0 logic. Semnalul **activare** este utilizat pentru activarea, respectiv dezactivarea unui numărător din calea de date. Acesta este un model foarte simplu de control al unui numărător din calea de date. Descrierea acestuia se face printr-un automat cu stări finite cu diagrama de stări din figura de mai jos.

Programul sursă VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity control_taste is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          clk : in  STD_LOGIC;
                              reset : in STD_LOGIC;
          activare : out  STD_LOGIC);
end control_taste;

architecture Behavioral of control_taste is

        type tip_stare is (st1, st2, st3);
        signal st: tip_stare;
begin
        process(clk, reset)
        begin
                if (reset = '0') then activare <= '0';
                elsif (clk'event and clk='1') then
                        case st is
                                when st1 =>
                                        if (a='0') then     st <= st2;
                                        elsif (b='0') then st <= st3;
                                        else      st <= st1;
                                        end if;
                                when st2 =>
                                        activare <= '1';
                                        st <= st1;
                                when st3 =>
                                        activare <= '0';
                                        st <= st1;
                                when others => null;
                        end case;
                end if;
        end process;
end Behavioral;
```

Principiul de funcționare al acestuia este asemănător cu modulele de același tip descrise în lucrarile anterioare. Este format doar dintr-un proces care se activează doar pe frontul pozitiv al semnalului de ceas. Semnalul de **reset** va plasa valoarea logică a semnalului activare în 0. Din starea **st1** se iese dacă unul din semnalele start sau stop sunt activate. Activarea concomitentă a acestora nu este

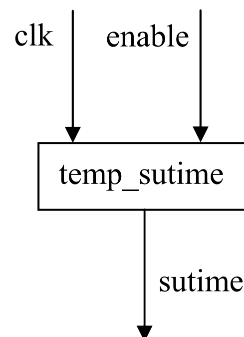
posibilă pentru că prima dată este tratată apăsarea tastei de start după care a tastei de stop.

Modulul temp_sutime

Acest modul este utilizat în divizarea frecvenței a semnalului **clk** de la cuarț de 50Mhz la 200Hz. Intern, este descris un numărător care are un registru cu valoarea de divizare calculată în formula de mai jos:

$$temp_num = \frac{50 \cdot 10^6 Hz}{200 Hz} = 25 \cdot 10^4 = 3D090_{(16)}$$

Este creat un fişier VHDL nou cu numele **temp_sutime**. Codul sursă al acestuia este următorul:




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity temp_sutime is
   Port (  clk : in  STD_LOGIC;
           enable: in STD_LOGIC;
           sutime : out  STD_LOGIC);
end temp_sutime;

architecture Behavioral of temp_sutime is
begin
        process(clk)
                variable temp_sutime: std_logic_vector(17 downto 0);
        begin
                if (clk'event and clk='1') then
                 if (enable = '1') then
                        sutime <= '0';
                        if (temp_sutime=x"3d08f") then
                                temp_sutime :=(others => '0');
                                sutime <= '1';
                        else
                                temp_sutime := temp_sutime + '1';
                        end if;
                 end if;
                end if;
        end process;
end Behavioral;
```

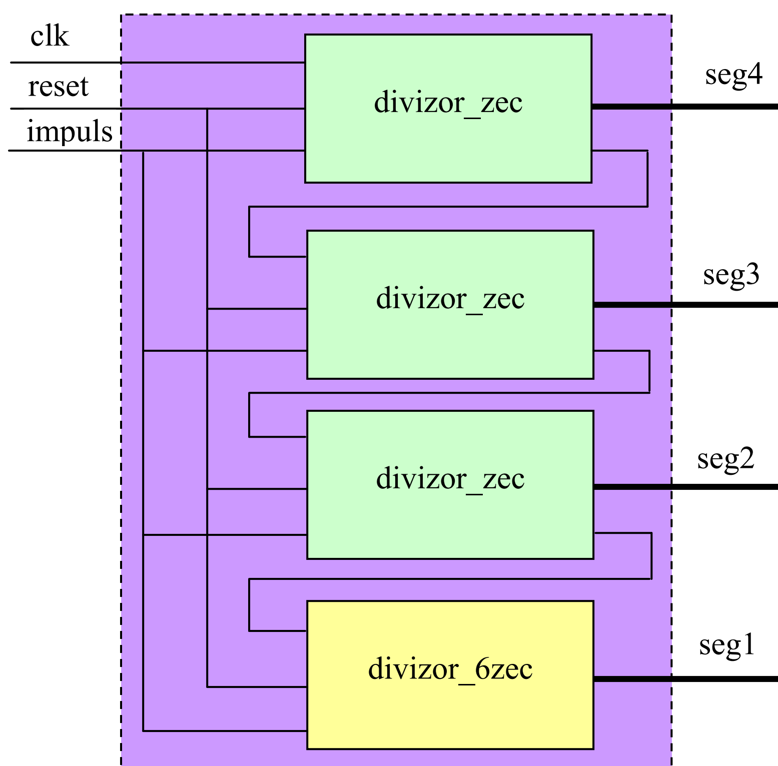
Descrierea funcțională a acestui divizor de ceas s-a făcut printr-un proces care se activează la semnalul **clk**. Este declarată o variabilă pe post de constantă de divizare cu o lățime de 18 biți şi o variabilă *temp_sutime*. Atunci când variabila ajunge la valoarea maximă dată, pe semnalul de ieşire se trimite un strob (o jumătate de perioadă a semnalului de ceas). S-a obținut frecvența de 200Hz pentru că ultima cifră a cronometrului trebuie să contorizeze sutimea de secundă şi va fi incrementată numai pe frontul pozitiv al semnalului **clk_out,** practic se incrementează cu o frecvență de 100Hz.

Modulul numărător

Acest modul este realizat pentru generarea sumelor de secundă, zecimilor de secundă, secunda sub formă binară, date ce vor fi preluate apoi de către decodoarele binar-7 segmente. Acest modul este rezultatul descrierii structurale a două componente:

- divizor_zec;
- divizor_6zec.

Schema de interconectare a acestora este dată în figura de mai jos:



Se creează un fişier VHDL cu numele *numarator* şi se introduce următorul program:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity numarator is
   Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          poarta : in  STD_LOGIC;
          seg1 : out  STD_LOGIC_VECTOR (3 downto 0);
          seg2 : out  STD_LOGIC_VECTOR (3 downto 0);
          seg3 : out  STD_LOGIC_VECTOR (3 downto 0);
          seg4 : out  STD_LOGIC_VECTOR (3 downto 0));
end numarator;

architecture Behavioral of numarator is
            component divizor_zec is
                        Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          impuls : in  STD_LOGIC;
          depasire : out  STD_LOGIC;
          data_out : out  STD_LOGIC_VECTOR (3 downto 0));
          end component;

            component divizor_6zec is
                        Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          impuls : in  STD_LOGIC;
          data_out : out  STD_LOGIC_VECTOR (3 downto 0));
          end component;

            signal transp: std_logic_vector(2 downto 0);
begin

            U1: divizor_zec port map(clk, reset, poarta, transp(0), seg4);
            U2: divizor_zec port map(clk, reset, transp(0), transp(1), seg3);
            U3: divizor_zec port map(clk, reset, transp(1), transp(2), seg2);
            U4: divizor_6zec port map(clk, reset, transp(2), seg1);
end Behavioral;
```

Cele două module sunt descrise comportamental. Practic sunt nişte numărătoare, primul de tip BCD iar cel de-al doilea numără pana la 6 după care se resetează.

Programul VHDL al modulului **divizor_zec** este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity divizor_zec is
   Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          impuls : in  STD_LOGIC;
          depasire : out  STD_LOGIC;
          data_out : out  STD_LOGIC_VECTOR (3 downto 0));
end divizor_zec;

architecture Behavioral of divizor_zec is

begin
        process(clk, reset)
                variable temp: std_logic_vector(3 downto 0);
        begin

                if reset='0' then temp:=(others=>'0');
                elsif (clk'event and clk='1') then
                   depasire <= '0';
                        if (impuls='1') then
                                if (temp="1001") then
                                        temp := "0000";
                                        depasire <= '1';
                                else
                                        temp := temp+1;
                                end if;
                        end if;
                end if;
        data_out <= temp;
        end process;

end Behavioral;
```

Modulul este sincronizat după semnalul global **clk**. Practic, semnalul **impuls** este doar de tipul *clock enable*. Când semnalul impuls este activ, modulul zecimal se incrementează cu o unitate. Incrementarea nu este continuă datorită faptului că semnalul **impuls** este doar o jumătate de perioadă de ceas.

Modulul divizor_6zec este asemănător cu cel de mai sus cu excepția faptului că nu mai are un semnal de ieşire pentru anunțarea depăşirii numărării şi condiția de depăşire este 6.

Programul VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity divizor_6zec is
   Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          impuls : in  STD_LOGIC;
          data_out : out  STD_LOGIC_VECTOR (3 downto 0));
end divizor_6zec;

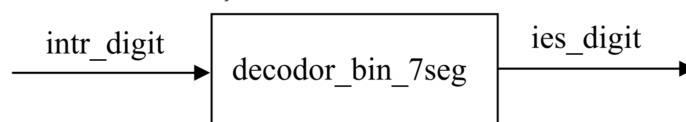
architecture Behavioral of divizor_6zec is

begin
        process(clk, reset)
                variable temp: std_logic_vector(3 downto 0);
        begin
                if reset='0' then temp:=(others=>'0');
                elsif (clk'event and clk='1') then
                        if (impuls='1') then
                                if (temp="0011") then
                                        temp := "0000";
                                else
                                        temp := temp+1;
                                end if;
                        end if;
                end if;
        data_out <= temp;
        end process;

end Behavioral;
```

Modul decodor_bin_7seg

Acest modul, ultimul din calea de date este utilizat pentru transformarea valorilor binare date de către numărător în cifre afişabile pe digiţii cu 7 segmente. Acest modul este descris printr-o specificaţie concurentă care asociază unui număr binar o valoare care corespunde afişării cifrei corespunzătoare. Modulul digital are următoarele porturi de intrare/ieşire.



Programul VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity decodor_bin_7seg is
        port(
                intr_digit : in STD_LOGIC_VECTOR(3 downto 0);
                ies_digit : out STD_LOGIC_VECTOR(6 downto 0));
end decodor_bin_7seg;

architecture Behavioral of decodor_bin_7seg is

begin
        with intr_digit select
        ies_digit <= "0111111" when "0000",
                             "0000110" when "0001",
                             "1011011" when "0010",
                             "1001111" when "0011",
                             "1100110" when "0100",
                             "1101101" when "0101",
                             "1111101" when "0110",
                             "0000111" when "0111",
                             "1111111" when "1000",
                             "1101111" when "1001",
                             "ZZZZZZZ" when others;

end Behavioral;
```

Toate aceste module sunt instațiate structural prin intermediului programului VHDL principal:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity cronometru is
   Port ( start : in  STD_LOGIC;
         stop : in  STD_LOGIC;
                    clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;
                    digit1 : out  STD_LOGIC_VECTOR (6 downto 0);
```

```
        digit2 : out  STD_LOGIC_VECTOR (6 downto 0);
        digit3 : out  STD_LOGIC_VECTOR (6 downto 0);
        digit4 : out  STD_LOGIC_VECTOR (6 downto 0));
end cronometru;

architecture Behavioral of cronometru is

        component control_taste is
   Port ( a : in  STD_LOGIC;
        b : in  STD_LOGIC;
        clk : in  STD_LOGIC;
                        reset : in STD_LOGIC;
        activare : out  STD_LOGIC);
        end component;

        component decodor_bin_7seg is
                port(
                        intr_digit : in STD_LOGIC_VECTOR(3 downto 0);
                        ies_digit  :  out  STD_LOGIC_VECTOR(6  downto
0));
        end component;

        component numarator is
   Port ( clk : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        poarta : in  STD_LOGIC;
        seg1 : out  STD_LOGIC_VECTOR (3 downto 0);
        seg2 : out  STD_LOGIC_VECTOR (3 downto 0);
        seg3 : out  STD_LOGIC_VECTOR (3 downto 0);
        seg4 : out  STD_LOGIC_VECTOR (3 downto 0));
        end component;

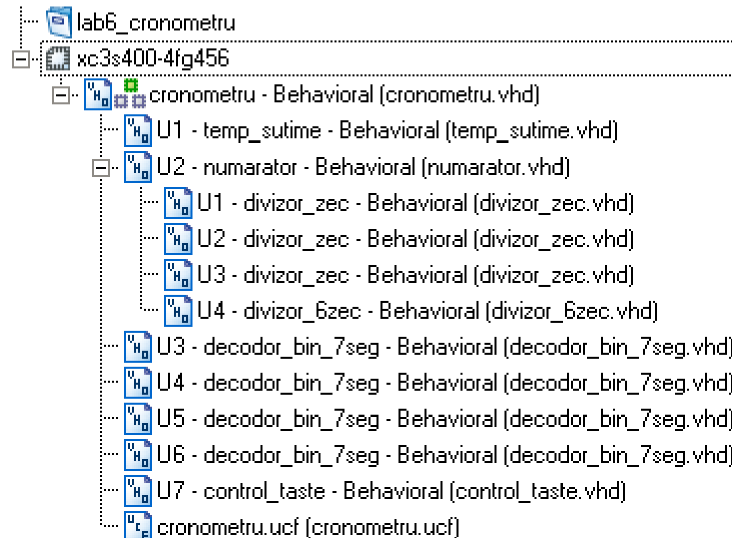
        component temp_sutime is
   Port ( clk : in  STD_LOGIC;
                        enable : in STD_LOGIC;
        sutime : out  STD_LOGIC);
        end component;

        signal tmp_sutime, temp_activ:std_logic;
        signal    temp_bin1,    temp_bin2,    temp_bin3,    temp_bin4:
std_logic_vector(3 downto 0);

begin
        U1: temp_sutime port map(clk, temp_activ, tmp_sutime);
        U2:  numarator  port  map(clk,  reset,  tmp_sutime,  temp_bin1,
temp_bin2, temp_bin3, temp_bin4);
        U3: decodor_bin_7seg port map(temp_bin1, digit1);
        U4: decodor_bin_7seg port map(temp_bin2, digit2);
```

```
        U5: decodor_bin_7seg port map(temp_bin3, digit3);
        U6: decodor_bin_7seg port map(temp_bin4, digit4);
        U7: control_taste port map(start, stop, clk, reset, temp_activ);
end Behavioral;
```

Structura de fişiere a proiectului este următoarea:



3. Exerciții

1. Să se verifice şi implementeze modulul digital descris pe parcursul lucrării de laborator;
2. Sa se modifice cronometrul modulul digital astfel încât să cronometreze continuu fără a mai nevoie de butoanele START şi STOP ;
3. Proiectați un ceas digital cu afişarea timpului pe 6 segmente (ore, minute, secunde). Realizați pentru acesta un fişier de testare completă.
4. Îmbunătățiți ceasul digital de la punctul anterior prin adăugarea a 6 butoane pentru reglarea timpului. Un buton va corespunde unui digit. La apăsarea butonului se va incrementa digitul respectiv cu o unitate zecimala. Validarea reglării se va face cu un buton suplimentar.
5. Propuneți şi implementați un automat care gestionează funcționarea corectă a patru semafoare într-o intersecție.

Capitolul 7
Utilizarea subprogramelor şi package-urilor în limbajul VHDL

În acest capitol sunt prezentate elemente de modularizare şi partiţionare a proiectului prin subprograme şi package-uri. Subprogramele constă în funcţii şi proceduri care sunt utilizate pentru operaţii comune. Package-ul este un mecanism prin care pot fi apelate subprograme sau module cu ajutorul librăriilor în cadrul unei entităţi. Subprogramele, tipurile şi componentele sunt elemente care pot fi unite în cadrul unei librării prin intermediul package-ului.

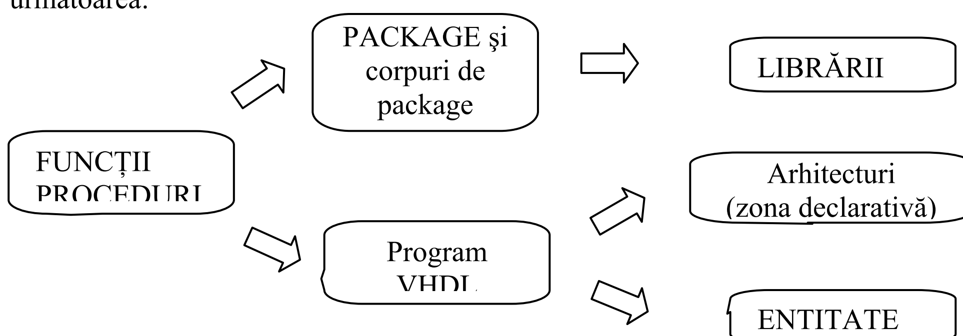
1. Breviar teoretic

Funcţiile şi procedurile sunt găsite sub denumirea de subprograme iar implementarea acestora se realizează la fel ca în limbajele software de nivel înalt C sau Pascal. O procedură poate returna unu sau mai multe argumente în timp ce funcţia returnează doar un singur argument. Într-o funcţie toţi parametrii sunt de intrare, în timp ce într-o procedură aceştia pot fi de intrare, ieşire sau de intrare/ieşire.

Funcţiile şi procedurile pot fi plasate în domeniul concurent sau în domeniul secvenţial. În domeniul concurent funcţiile şi procedurile sunt plasate în afara proceselor sau a altor subprograme. În schimb, funcţiile şi procedurile secvenţiale există numai în interiorul proceselor sau altor subprograme şi toate specificaţiile conţinute de acestea sunt de asemenea secvenţiale.

O procedură este declarată ca o specificaţie separată într-un proces sau arhitectură, în timp ce o funcţie este utilizată ca o specificaţie de atribuire sau o expresie.

Plasarea funcţiilor şi procedurilor în cadrul programelor VHDL este următoarea:



1.1. Funcții şi proceduri

Întotdeauna sunt două clase mari de funcții pentru utilizarea variabilelor în VHDL: *funcții de conversie şi funcții de rezoluție*.

a.Funcțiile de conversie:

Funcțiile de conversie sunt utilizate în translatarea unui obiect de un tip în altul. Acestea sunt utilizate în specificațiile de instanțiere a componentelor pentru a da posibilitatea interconectării de semnale şi porturi de tipuri diferite. Situații care apar atunci când proiectantul conectează module din anumite proiecte care au caracteristicile porturilor diferite.

b.Funcții de rezoluție

O funcție de tip rezoluție este utilizată la atribuirea unei singure stări la un semnal care este condus de mai multe drivere. În VHDL nu este recomandată utilizarea semnalelor de tip multisursă (driver multiplu).

Când o funcție de tip rezoluție este executată, returnează o singură valoare rezultată de la toate valorile logice ale driverelor. Această stare va fi valoarea nouă a semnalului numită „resolved value".

1.2. PACKAGE-uri

Prin experiența acumulată anterior în realizarea proiectelor cu limbajul VHDL se observă că modulele digitale sunt scrise în general sub formă de componente, funcții sau proceduri.
Toate acestea pot fi plasate în PACKAGE-uri şi compilate în cadrul unei librării (LIBRARY) după cum se observă în figura de mai jos. Prin această tehnică este creată posibilitatea partiționării, apelării şi reutilizării codului VHDL pe subprograme.
Sintaxa unei structuri de tip PACKAGE este următoarea:

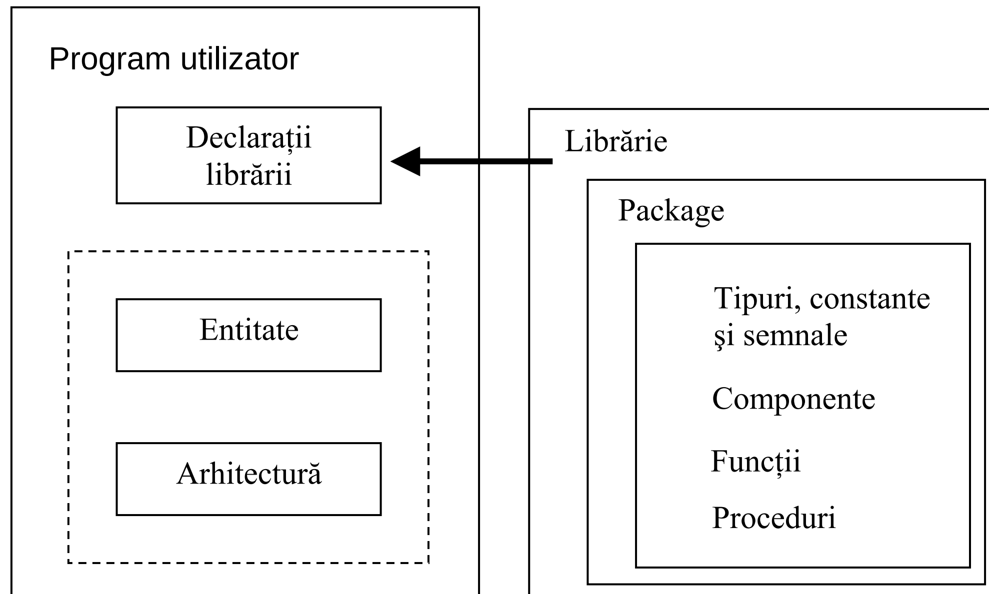
```
PACKAGE nume_package IS
(declarații)
END nume_package

[PACKAGE BODY nume_package IS
(descrieri de funcții şi proceduri)
END nume_package;]
```

Se observă că sintaxa este compusă din două corpuri:

- PACKAGE în care se găsesc declarațiile de componente, funcții, proceduri, tipuri, constante, etc.

- PACKAGE BODY în care sunt făcute descrierile funcțiilor și procedurile anunțate în zona declarativă.



2. Aplicații

a) Să se realizeze o funcție de detecție a frontului crescător în limbajul VHDL.

Soluție

În cadrul programului VHDL următor este realizată în zona declarativă a arhitecturii o funcție prin care este determinat frontul crescător, respectiv descrescător al unui semnal.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity bist_D_Functie is
        port(
                d : in STD_LOGIC;
                clk : in STD_LOGIC;
                rst : in STD_LOGIC;
                q : out STD_LOGIC
                );
end bist_D_Functie;

architecture bist_D_Functie of bist_D_Functie is
```

```
		function front_crescator(signal fc: std_logic)
			return boolean is
		begin
			return (fc'event and fc='1' and fc'last_value='0');
		end function;
	begin
		process (rst,clk)
		begin
			if (rst='1') then q<='0';
			elsif front_crescator(clk) then
				q<=d;
			end if;
		end process;
	end bist_D_Functie;
```

Funcția *front_crescător* este definită în zona declarativă a arhitecturii. Are ca argument de intrare un semnal de tip **std_logic** şi returnează o valoare booleană. In funcție se testează dacă a apărut un eveniment pe semnal iar valoarea logică după apariția evenimentului este 1 şi valoarea anterioară a fost 0.

Apelarea acestei funcții se face în cadrul unui proces, front_crescator(clk), pentru determinarea frontului pozitiv a semnalului de ceas.

În multe cazuri, funcțiile sunt utilizate pentru returnarea unei valori în cadrul unei expresii.

b) Să se descrie un numărător creat cu obiecte definite în librăria **IEEE STD 1076** şi apoi aplicat pe un modul care lucrează în standardul **std_logic_1164** pentru a putea fi implementat în FPGA.

Soluție

În descrierea acestei aplicații sunt realizați următorii paşi:

- Se creează un proiect gol cu numele **num_std** ;
- Se descrie modulul *numărător* prin standardul 1076 în care nu există tipul **std_logic**, subpunctul a. Sunt folosite tipurile **bit** şi **integer**;
- Sunt realizate funcții de conversie din numere întregi în **std_logic** şi din **std_logic** în **bit**.
- Se implementează modulul propriu zis de numărare cu porturile de tip **std_logic**, subpunctul b.

a) Implementarea numărătorului

Se creează un *fişier vhd* cu numele *numarator_integer.vhd* în care se introduc următoarele linii de cod:

```
LIBRARY IEEE;
ENTITY numarator_integer IS
	PORT(
		clk : in bit;
```

```
                data_out : out integer range 0 to 10);
END numarator;

ARCHITECTURE numarator_integer OF numarator_integer IS
BEGIN
        PROCESS (clk)
                VARIABLE temp_num: integer range 0 to 10;
        BEGIN
                IF (clk'event AND clk='1')THEN
                        temp_num:=temp_num+1;
                        IF (temp_num=10) THEN      temp_num:=0;
                        END IF;
                END IF;

                data_out<=temp_num;
        END PROCESS;
END numarator_integer;
```

Numărarea se face în cadrul unui proces prin incrementarea variabilei *temp_num*. În finalul procesului, semnalul **data_out** preia conținutul variabilei *temp_num*.

În cadrul procesului nu poate fi realizată direct incrementarea semnalului pentru că apoi se testează dacă aceasta a ajuns la valoarea 10 în zecimal. Actualizarea valorii unui semnal se face numai atunci când se închide procesul şi practic testul se realiza asupra valorii pe care o avea semnalul la intrarea în proces.

b) Implementarea funcțiilor şi programului principal

Se creează în acelaşi proiect un fişier VHDL cu numele *numarator_std.vhd* care va conține următoarele linii de cod:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity num_std is
        port(
                ceas : in STD_LOGIC;
                iesire : out STD_LOGIC_VECTOR(3 DOWNTO 0)  );
end num_std;

architecture num_std of num_std is

subtype my_integer is integer range 0 to 10;

        function stdlogic_in_bit(signal s: std_logic)return bit is
                variable rezultat: bit;
        begin
                if s='1' then rezultat:='1';
                else rezultat:='0';
```

```
		end if;
		return rezultat;
	end function;

	function intreg_in_stdlogic(s: my_integer) return STD_LOGIC_VECTOR is
		variable resultat: STD_LOGIC_VECTOR (3 downto 0);
		variable temp: my_integer;
	begin
		temp := s;
		for i in 0 to 3 loop
			if (temp mod 2) = 1 then
				resultat(i) := '1';
			else
				resultat(i) := '0';
			end if;
			if temp > 0 then
				temp := temp / 2;
				else
					temp := (temp - 1) / 2;
				end if;
			end loop;
			return resultat;
		end function;

		component numarator_integer is
			port(	clk : in bit;
				data_out : out my_integer);
		end component;

		signal clk1:bit;
		signal iesire_temp : my_integer;

	begin
		 clk1<= stdlogic_in_bit(ceas);
	    U1: numarator_integer port map(clk => clk1,data_out => iesire_temp);
		 iesire <= intreg_in_stdlogic(iesire_temp);
	end num_std;
```

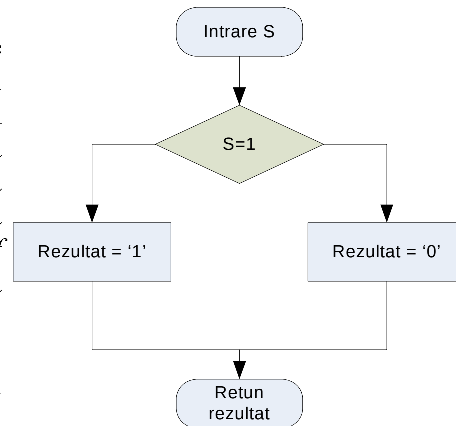
Modulul numărător este realizat în standardul 1073 şi are porturile I/O de tip *bit* sau *integer*. Dacă dorim să-l implementăm într-o structură reconfigurabilă, aceste porturi nu sunt compatibile cu formatul digital *std_logic*. În acest caz, este necesară transformarea acestora din *std_logic* în *bit* şi din *integer* în *std_logic_vector*.

Pentru realizarea transformărilor amintite, sunt implementate două funcții:

- function stdlogic_in_bit(signal s: std_logic)return bit prin care se face trecerea din **std_logic** în **bit**;

- function intreg_in_stdlogic(s: my_integer) return STD_LOGIC_VECTOR utilizată pentru conversia din **integer** în **std_logic_vector**.

Prima funcție are ca parametru de intrare semnalul **s** de tip *std_logic* şi returnează o valorare de tipul *bit*. În interiorul acesteia este declarată o variabilă cu numele rezultat de tipul *bit*. Această variabilă va prelua valoarea logică a semnalului de intrare prin specificația *if* apoi este returnată funcției prin specificația *return*.



Schema logică a funcției stdlogic_in_bit este dată în figura alăturată.

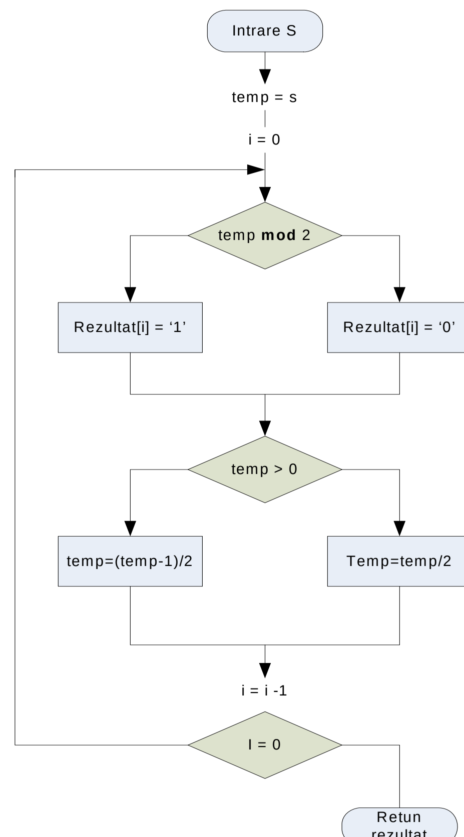
A doua funcție are ca parametru de intrare semnalul **s** de tip *integer* şi returnează o valorare de tipul *std_logic_vector*.

Observăm în relația de mai jos că:

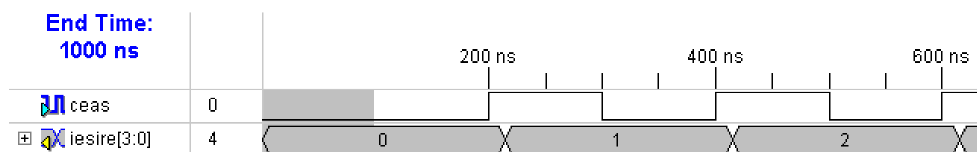
$$temp = 2^3 \cdot rez[3] + 2^2 \cdot rez[2] + 2^1 \cdot rez[1] + rez[0]$$

Algoritmul de conversie din **integer** în **std_logic_vector** se bazează pe metoda tradițională de transformare a unui număr zecimal în formă binară.

Prin testul *temp mod 2* este determinat modulo 2. Apoi se face împărțirea la 2 a variabilei 2.



Simularea funcțională a modului digital prezentat anterior este dată în graficul de mai jos:



c) Să se realizeze o funcție rezoluție pentru tipul **patru_stări**.

```
TYPE patru_stari IS (X, 1, 0, Z);
```

Soluție

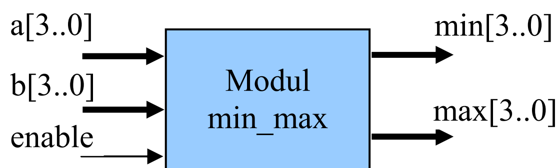
În cadrul acestui tip sunt prezentate patru stări diferite posibile ce pot fi obținute pe un semnal: „1" logic, „0" logic, „Z" este înaltă impedanță și valoarea „X" care reprezintă condiție necunoscută. De exemplu rezultatul a două drivere cu stările logice „1" și „0" în tabelul de adevăr al funcției de rezoluție este:

	Z	0	1	X
Z	Z	0	1	X
0	0	0	X	X
1	1	X	1	X
X	X	X	X	X

Acest tabel este realizat doar pentru două drivere și se găsește semnalul rezultat prin aplicarea acestora în același timp.

d) Să se descrie un modul digital care determină minimul și maximul dintre două numere reprezentate pe câte un octet. Funcția de minim este realizată în cadrul unei proceduri.

Entitatea modulului digital este următoarea:



Determinarea minimului si maximului se face prin următorul cod VHDL:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.all;

entity modul_min_max is
	port(
		enable : in STD_LOGIC;
		a : in STD_LOGIC_VECTOR(3 downto 0);
		b : in STD_LOGIC_VECTOR(3 downto 0);
		min : out STD_LOGIC_VECTOR(3 downto 0);
		max : out STD_LOGIC_VECTOR(3 downto 0)
		);
end modul_min_max;

architecture modul_min_max of modul_min_max is

	subtype my_natural is integer range 0 to 15;

	procedure min_max(x,y:in my_natural;
		min,max:out my_natural) is
	begin
		if (x<y) then
			min :=x;
			max :=y;
		else
			min :=y;
			max :=x;
		end if;
	end min_max;

	function   std_la_intreg(s:std_logic_vector(3   downto   0))   return
my_natural is
		variable rezultat: my_natural;
	begin
		rezultat := 0;
		for i in 3 downto 0 loop
			rezultat := rezultat * 2;
			if s(i)='1'then
				rezultat := rezultat+ 1;
			end if;
		end loop;
		return rezultat;
	end function;

function intreg_la_std(s: my_natural) return STD_LOGIC_VECTOR is
		variable resultat: STD_LOGIC_VECTOR (3 downto 0);
		variable temp: my_natural;
	begin
		temp := s;
		for i in 0 to 3 loop
```

```
                if (temp mod 2) = 1 then
                    resultat(i) := '1';
                else
                    resultat(i) := '0';
                end if;
                if temp > 0 then
                    temp := temp / 2;
                else
                    temp := (temp - 1) / 2;
                end if;
            end loop;
            return resultat;
        end function;

begin

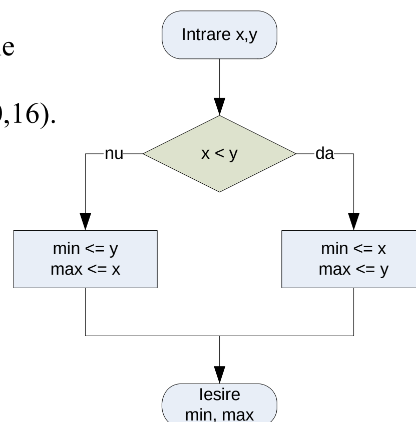
        process(enable,a,b)
        variable temp1, temp2:my_natural;
        begin
            if (enable ='0') then

        min_max(std_la_intreg(a),std_la_intreg(b),temp1,temp2);
                min <= intreg_la_std(temp1);
                max <= intreg_la_std(temp2);
            end if;
        end process;
end modul_min_max;
```

Programul VHDL conținte o procedură şi două funcții prin care sunt realizate următoarele operații: determinarea minimului şi maximului, conversia din întreg în std_logic_vector şi conversia din std_logic_vector în întreg.

a) Procedura min_max

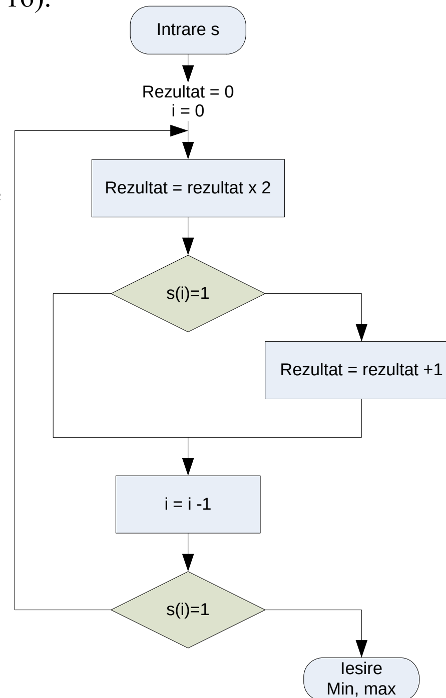
Are ca argumente de intrare numerele *x* şi *y* iar ca valori de ieşire *min* şi *max*. Toate numerele sunt naturale pe intervalul [0,16).



Algoritmul de determinare a minimului, respectiv maximului este dat schema logică alăturată.

b) **Funcția std_la_intreg**

Această funcție are ca argument de intrare *std_logic_vector* pe 4 biți și returnează o valoare naturală pe intervalul [0, 16).



Metoda de conversie se bazează pe formula:

$$rezultat = 2^3 \cdot s(3) + 2^2 \cdot s(2) + 2^1 \cdot s(1) + s(0)$$
$$= 2(2(2 \cdot s(3) + s(2)) + s(1)) + s(0)$$

Algoritmul de implementare a formulei de mai sus este dat în schema logică alăturată.

c) **Funcția intreg_la_std**

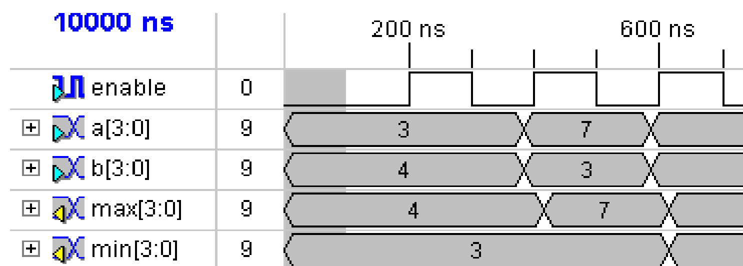
Are ca argument de intrare o variabilă de tip natural definită pe intervalul [0, 16) și returnează un vector de tip std_logic pe 4 biți.

Algoritmul după care s-a realizat implementarea acesteia este același cu cel de la exemplul anterior.

Corpul arhitecturii este compus din specificația **if** prin care se testează dacă a fost activat semnalul **enable** și apelarea procedurii *min_max*.

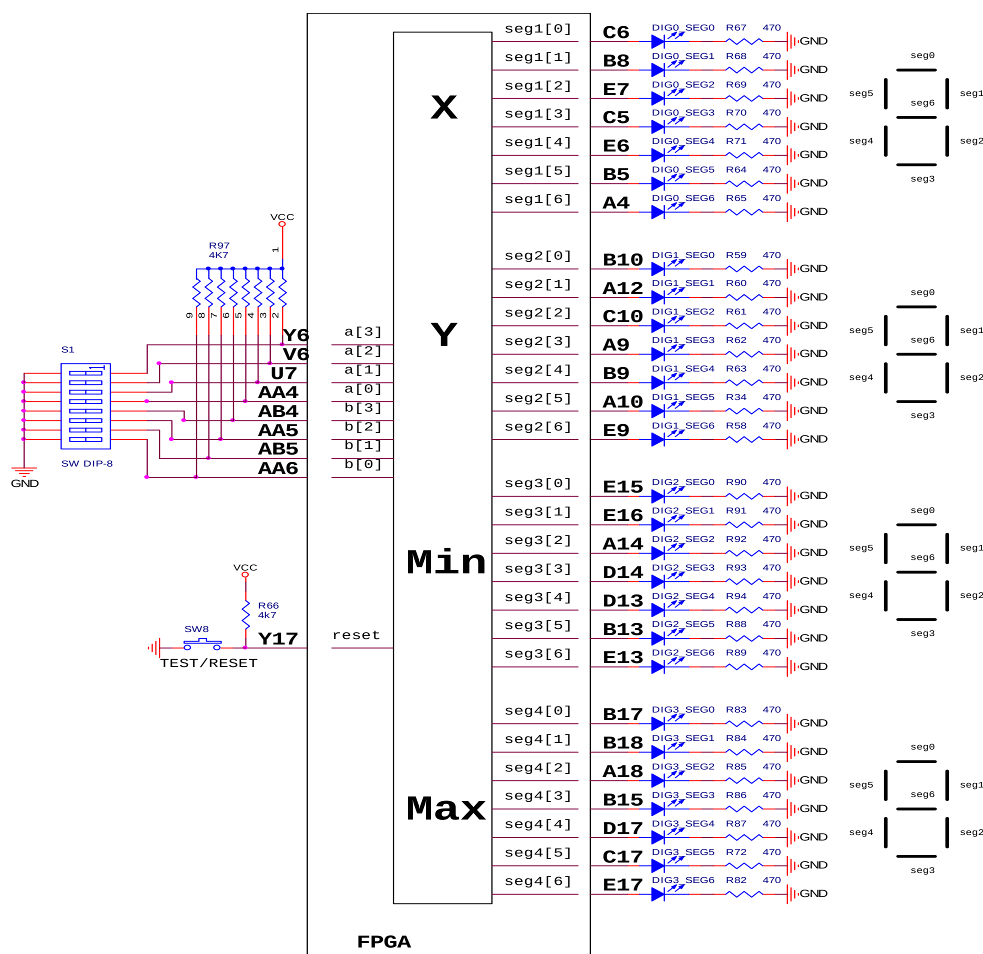
Din cauză că argumentele de intrare-ieșire ale procedurii *min_max* sunt de tip natural, sunt folosite funcțiile *std_la_intreg* și *intreg_la_std* pentru conversia corespunzătoare a semnalelor.

Simularea funcțională a modulului prezentat anterior este dată în figura de mai jos:



e) Funcții şi Proceduri în PACKAGE

În această aplicație este preluat exemplul anterior dar cu implementarea funcțiilor şi procedurilor în cadrul unui package. În plus, se cere afişarea operatorilor de intrare, respectiv minimului şi maximului pe câte un digit după cum se arată în schema de mai jos.



a) în cadrul unui proiect nou cu numele *min_max* se creează un fişier de tip VHDL cu numele *librarie_min_nax.vhd* în care este introdus codul de mai jos.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
```

```
package my_package is
	subtype my_natural is integer range 0 to 15;
	procedure min_max(x,y:in my_natural; min,max:out my_natural);
	function std_la_intreg(s:std_logic_vector(3 downto 0)) return my_natural;
	function intreg_la_std(s: my_natural) return STD_LOGIC_VECTOR;
	function conv_std_7seg(s:std_logic_vector(3 downto 0)) return
std_logic_vector;
end my_package;


package body my_package is

	procedure min_max(x,y:in my_natural;
		min,max:out my_natural) is
	begin
		if (x<y) then
			min :=x;
			max :=y;
		else
			min :=y;
			max :=x;
		end if;
	end min_max;

	function std_la_intreg(s:std_logic_vector(3 downto 0)) return my_natural is
		variable rezultat: my_natural;
	begin
		rezultat := 0;
		for i in 3 downto 0 loop
			rezultat := rezultat * 2;
			if s(i)='1'then
				rezultat := rezultat+ 1;
			end if;
		end loop;
		return rezultat;
	end function;

	function intreg_la_std(s: my_natural) return STD_LOGIC_VECTOR is
		variable resultat: STD_LOGIC_VECTOR (3 downto 0);
		variable temp: my_natural;
	begin
		temp := s;
		for i in 0 to 3 loop
			if (temp mod 2) = 1 then
				resultat(i) := '1';
			else
				resultat(i) := '0';
			end if;
```

```
                if temp > 0 then
                        temp := temp / 2;
                else
                        temp := (temp - 1) / 2;
                end if;
        end loop;
        return resultat;
    end function;

    function    conv_std_7seg(s:std_logic_vector(3    downto    0))    return
std_logic_vector is
        variable temporar:std_logic_vector(6 downto 0);
    begin
        case s is
                when "0000" => temporar :="0111111";
                when "0001" => temporar :="0000110";
                when "0010" => temporar :="1011011";
                when "0011" => temporar :="1001111";
                when "0100" => temporar :="1100110";
                when "0101" => temporar :="1101101";
                when "0110" => temporar :="1111101";
                when "0111" => temporar :="0000111";
                when "1000" => temporar :="1111111";
                when "1001" => temporar :="1101111";
                when "1010" => temporar :="1110111";
                when "1011" => temporar :="1111100";
                when "1100" => temporar :="0111001";
                when "1101" => temporar :="1110011";
                when "1110" => temporar :="1111001";
                when "1111" => temporar :="1110001";
                when others => null;
        end case;
        return temporar;
    end function;
end my_package;
```

Pachetul se numeşte my_package şi este format din două părţi:

- Partea declarativă. Sunt declarate funcțiile descrise în corpul package-ului şi subtipul natural care este definit pe domeniul 0..15 al numerelor întregi;
- În partea a doua, adică zona package body, sunt descrise funcțiile care vor fi utilizate în programul principal.

În plus, fața de exemplul anterior a fost introdusă *funcția std_7seg* care face conversia binară a unui semnal pe 4 biți de tip **std_logic** în 7 segmente pentru afişarea pe digiți cu 7 segmente.

Decodarea se face pe baza unei tabele de corespondență între valoarea binară şi valoarea care trimisă către afişorul cu 7 segmente pentru a afişa cifra corespunzătoare.

b) În cadrul aceluiaşi proiect se creează un al fişier de tip VHDL cu numele *min_max.vhd* şi se introduce următorul cod:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use my_package.all;

entity modul_min_max is
	port(
		enable : in STD_LOGIC;
		a : in STD_LOGIC_VECTOR(3 downto 0);
		b : in STD_LOGIC_VECTOR(3 downto 0);
		seg1, seg2 : out STD_LOGIC_VECTOR(6 downto 0);
		seg3, seg4 : out STD_LOGIC_VECTOR(6 downto 0));
end modul_min_max;

architecture modul_min_max of modul_min_max is
	signal min,max:std_logic_vector(3 downto 0);
begin

	process(enable)
		variable temp1, temp2:my_natural;
	begin
		if (enable ='0') then
			min_max(std_la_intreg(a),std_la_intreg(b),temp1,temp2);
			min <= intreg_la_std(temp1);
			max <= intreg_la_std(temp2);
		end if;
	end process;

	process(a,b,min,max)
	begin
		seg1 <= conv_std_7seg(a);
		seg2 <= conv_std_7seg(b);
		seg3 <= conv_std_7seg(min);
		seg4 <= conv_std_7seg(max);
	end process;
end modul_min_max;
```

Codul din fişierul de mai sus reprezintă programul principal. În zona declarativă a librăriilor a fost introdusă linia

```
use my_package.all;
```

prin care specifică compilatorului să introducă toate funcțiile din package-ul *my_package* creat în fişierul *librărie_min_max.vhd*.

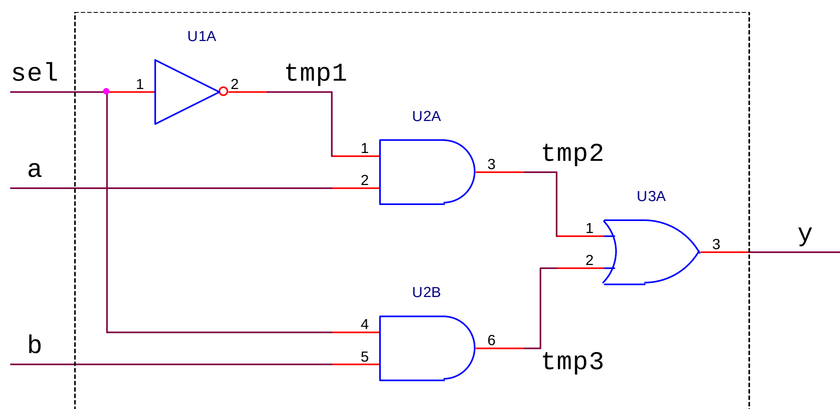
Programul principal este format din două procese. Primul proces realizează operaţia de aflare a minimului atunci când este activat semnalul **enable** iar cel de-al doilea realizează conversia din binar în 7 segmente prin funcţia *conv_std_7seg*.

c) Implementarea

- se compilează proiectul şi se verifică dacă apar erori;
- se va face implementarea la fel ca şi la proiectele anterioare prin indicarea căii unde este găsit fişierul cu extensia *ucf* şi activării generării fişierului de configurare cu extensia *bit*.

f) Componente în PACKAGE

Este prezentată implementarea modulului digital din figura de mai jos. În acest caz, componentele (inversorul, poarta SI şi poarta SAU) vor fi preluate dintr-un PACKAGE creat de programator.



a) Se creează un proiect nou
b) Se creează un fişier gol VHDL cu numele *componenete.vhd* având codul de mai jos şi se ataşează la proiect.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity inversor is
        port(
```

```
            in1 : in STD_LOGIC;
            ies : out STD_LOGIC
            );
end inversor;

architecture inversor of inversor is
begin
      ies <= not in1;
end inversor;

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity poarta_si is
      port(
            in1 : in STD_LOGIC;
            in2 : in STD_LOGIC;
            ies : out STD_LOGIC
            );
end poarta_si;

architecture poarta_si of poarta_si is
begin
      ies <= in1 and in2;
end poarta_si;

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity poarta_sau is
      port(
            in1 : in STD_LOGIC;
            in2 : in STD_LOGIC;
            ies : out STD_LOGIC
            );
end poarta_sau;

architecture poarta_sau of poarta_sau is
begin
      ies <= in1 or in2;
end poarta_sau;
```

În programul de mai sus, prin descriere de tip flux de date sunt create componentele *inversor*, *poarta_şi* şi *poarta_sau*.

c) Se ataşează un nou fişier VHDL cu numele componente_package.vhd conținând codul VHDL:

```
package componente is

component inversor is
        port(
                in1 : in STD_LOGIC;
                ies : out STD_LOGIC
        );
end component;

component poarta_si is
        port(
                in1 : in STD_LOGIC;
                in2 : in STD_LOGIC;
                ies : out STD_LOGIC
        );
end component;

component poarta_sau is
        port(
                in1 : in STD_LOGIC;
                in2 : in STD_LOGIC;
                ies : out STD_LOGIC
        );
end component;

end componente;
```

Practic, acesta este fişierul în care sunt ataşate componentele la package-ul creat de utilizator. Diferența față de exemplul anterior este că acest package nu prezintă package_body. Comportamentul componentelor declarate în acest package a fost descris în fişierul creat anterior.

d) Se creează programul principal numit *componente_main.vhd* care se ataşează la proiect. Codul VHDL este următorul:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use componente.all;

entity componente_main is
        port(
                a : in STD_LOGIC;
                b : in STD_LOGIC;
                sel : in STD_LOGIC;
```

```
                y : out STD_LOGIC
            );
end componente_main;

architecture componente_main of componente_main is
signal tmp1, tmp2, tmp3: std_logic;
begin
        U1A: inversor port map(sel,tmp1);
        U2A: poarta_si port map(tmp1, a, tmp2);
        U2B: poarta_si port map(sel, b, tmp3);
        U3A: poarta_sau port map(tmp2, tmp3, y);
end componente_main;
```

În acest fişier se face o descriere structurală numai că, în exemplul de față, componentele sunt preluate din package-ul definit de utilizator componente.all şi nu sunt declarate în zona declarativă a arhitecturii.

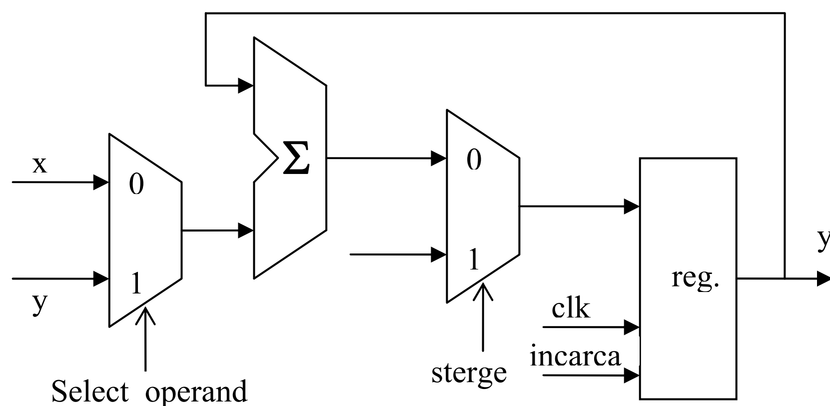
3. Exerciții

1. Implementați programele VHDL din cadrul lucrării;

2. Propuneți alte fişiere de tipul test-bench pentru testarea completă a acestora;
3. Încercați să implementați un exemplu simplu din laboratoarele anterioare după care creați pentru acesta un fişier de test. Realizați testele pentru un circuit în condiții reale, după implementare, şi faceți o comparație cu rezultatele obținute în care nu se ținea cont de timpii de propagare. Analizați stările intermediare (de comutație a semnalelor).
4. Să se realizeze un sumator după figura de mai jos:

   Acesta deține doi operanzi de intrare **x** şi **y** reprezentați pe câte 8 biți. Selectarea lor se face prin semnalul **select_operand**. Setarea în zero a rezultatului se face prin intermediul semnalului **sterge**.

   Descrierea acestui modul se va face în următorul mod. Toate componentele se vor introduce într-un package creat de utilizator. Descrierea sumatorului se va face structural în programul principal utilizând componentele descrise în package.



BIBLIOGRAFIE

1. S. Brown, Z. Vranesic, Fundamentals of Digital Logic With VHDL Design, University of Toronto, McGraw Hill, 2005
2. D. L. Perry, VHDL Programming by Examle, McGraw Hill, 2002
3. J. Weber, M. Meaudre, Le langage VHDL Cours et exercises, Dumond, 2004
4. IEEE Computer Society, 1076 IEEE Standard VHDL – Language Reference Manual, IEEE, 2002
5. V. A. Pedroni, Circuit Design with VHDL, Cambrige, MIT Press, 2004
6. E. O. Hwang, Digital Logic and Microprocessor Design With VHDL, La Sierra University, Riverside, 2005
7. R. Dueck, Digital Design with CPLD Applications and VHDL, Hardcover, 2000
8. P.P.Chu, FPGA Prototyping by VHDL Examples – Xilinx Spartan 3 Version, WILEY, 2008

ANEXA 1

Corelarea dintre pinii componentelor de pe sistemul de dezvoltare Live Design cu circuitul SPARTAN3 XC3S400 utilizat în aplicațiile din această carte.

Digit 0	PIN
Dig0_seg0	**C6**
Dig0_seg1	**B8**
Dig0_seg2	**E7**
Dig0_seg3	**C5**
Dig0_seg4	**E6**
Dig0_seg5	**B5**
Dig0_seg6	**A4**
Dig0_seg7	**C7**

Digit 1	PIN
Dig1_seg0	**B10**
Dig1_seg1	**A12**
Dig1_seg2	**C10**
Dig1_seg3	**A9**
Dig1_seg4	**B9**
Dig1_seg5	**A10**
Dig1_seg6	**E9**
Dig1_seg7	**D11**

Digit 2	PIN
Dig2_seg0	**E15**
Dig2_seg1	**E16**
Dig2_seg2	**A14**
Dig2_seg3	**D14**
Dig2_seg4	**D13**
Dig2_seg5	**B13**
Dig2_seg6	**E13**
Dig2_seg7	**D15**

Digit 3	PIN
Dig3_seg0	**B17**
Dig3_seg1	**B18**
Dig3_seg2	**A18**
Dig3_seg3	**B15**
Dig3_seg4	**D17**
Dig3_seg5	**C17**
Dig3_seg6	**E17**
Dig3_seg7	**A19**

Digit 4	PIN
Dig4_seg0	**D19**
Dig4_seg1	**F18**
Dig4_seg2	**C20**
Dig4_seg3	**C19**
Dig4_seg4	**C18**
Dig4_seg5	**B19**
Dig4_seg6	**D18**
Dig4_seg7	**F17**

Digit 5	PIN
Dig5_seg0	**F19**
Dig5_seg1	**G17**
Dig5_seg2	**E19**
Dig5_seg3	**D21**
Dig5_seg4	**D20**
Dig5_seg5	**E18**
Dig5_seg6	**C22**
Dig5_seg7	**E20**

LEDURI

Nume led	PIN
LED0	**W2**
LED1	**Y1**
LED2	**Y2**
LED3	**Y3**
LED4	**W4**
LED5	**W5**
LED6	**Y5**
LED7	**W6**

TASTE

Nume_tasta	PIN
SW_USER0	**D1**
SW_USER1	**C1**
SW_USER2	**B6**
SW_USER3	**A15**
SW_USER4	**B20**
SW_USER5	**C21**
TEST/RESET	**Y17**

CLK **AA12**

SWITCH

Nume_linie	PIN
SW_DIP0	**Y6**
SW_DIP1	**V6**
SW_DIP2	**U7**
SW_DIP3	**AA4**
SW_DIP4	**AB4**
SW_DIP5	**AA5**
SW_DIP6	**AB5**
SW_DIP7	**AA6**

PORT RS232

Nume_linie	PIN
RS232_RTS	**E1**
RS232_TX	**F7**
RS232_RX	**A5**
RS232_CTS	**F2**

PC2 KEYBOARD

Nume_linie	PIN
KBDATA	**G19**
KBCLOCK	**F20**

PC2 MOUSE

Nume_linie	PIN
MOUSEDATA	**G18**
MOUSECLOCK	**L17**

VGA

Nume_linie	PIN
RED2	**D6**
RED1	**D7**
RED0	**D9**
GREEN2	**E11**
GREEN1	**C11**
GREEN0	**D10**
BLUE2	**E14**
BLUE1	**A13**
BLUE0	**C13**
HSYNC	**A8**
VSYNC	**B14**

AUDIO

Nume_linie	**PIN**
AUDIOL	**W3**
AUDIOR	**U3**

SRAM0

Nume_linie	**PIN**
SRAM0_D0	**L4**
SRAM0_D1	**L3**
SRAM0_D2	**M5**
SRAM0_D3	**M4**
SRAM0_D4	**M3**
SRAM0_D5	**N4**
SRAM0_D6	**N3**
SRAM0_D7	**T5**
SRAM0_D8	**T4**
SRAM0_D9	**T6**
SRAM0_D10	**M6**
SRAM0_D11	**N2**
SRAM0_D12	**N1**
SRAM0_D13	**M2**
SRAM0_D14	**M1**
SRAM0_D15	**L2**
SRAM0_A0	**L6**
SRAM0_A1	**K4**
SRAM0_A2	**H5**
SRAM0_A3	**G6**
SRAM0_A4	**F3**
SRAM0_A5	**G1**
SRAM0_A6	**G2**
SRAM0_A7	**K3**
SRAM0_A8	**T2**
SRAM0_A9	**T1**
SRAM0_A10	**U2**
SRAM0_A11	**V3**
SRAM0_A12	**V1**
SRAM0_A13	**W1**
SRAM0_A14	**V2**
SRAM0_A15	**V5**
SRAM0_A16	**V4**
SRAM0_A17	**U5**
SRAM0_A18	**U6**
SRAM0_OE	**K1**
SRAM0_UB	**K2**
SRAM0_LB	**L1**
SRAM0_WE	**U4**
SRAM0_E	**L5**

SRAM1

Nume_linie	**PIN**
SRAM1_D0	**L21**
SRAM1_D1	**M22**
SRAM1_D2	**M21**
SRAM1_D3	**N22**
SRAM1_D4	**N21**
SRAM1_D5	**U20**
SRAM1_D6	**T22**
SRAM1_D7	**T21**
SRAM1_D8	**V18**
SRAM1_D9	**U19**
SRAM1_D10	**U18**
SRAM1_D11	**T18**
SRAM1_D12	**R18**
SRAM1_D13	**T17**
SRAM1_D14	**M18**
SRAM1_D15	**M20**
SRAM1_A0	**K21**
SRAM1_A1	**K22**
SRAM1_A2	**K20**
SRAM1_A3	**G21**
SRAM1_A4	**G22**

SRAM1_A5	**M17**
SRAM1_A6	**L18**
SRAM1_A7	**K19**
SRAM1_A8	**V19**
SRAM1_A9	**W20**
SRAM1_A10	**W19**
SRAM1_A11	**Y20**
SRAM1_A12	**Y21**
SRAM1_A13	**Y22**
SRAM1_A14	**W21**
SRAM1_A15	**W22**
SRAM1_A16	**V21**
SRAM1_A17	**V22**
SRAM1_A18	**V20**
SRAM1_OE	**L19**
SRAM1_UB	**L20**
SRAM1_LB	**M19**
SRAM1_WE	**U21**
SRAM1_E	**L22**

HDR1

Nume_linie	**PIN**
IO0	**V7**
IO1	**AA8**
IO2	**AB8**
IO3	**V8**
IO4	**Y10**
IO5	**V9**
IO6	**W9**
IO7	**AA1**
IO8	**AB10**
IO9	**W10**
IO10	**AB11**
IO11	**U11**
IO12	**AB13**
IO13	**AA13**
IO14	**V10**
IO15	**U10**
IO16	**W13**
IO17	**Y13**

HDR2

Nume_linie	**PIN**
IO18	**V14**
IO19	**V13**
IO20	**AA15**
IO21	**W14**
IO22	**AB15**
IO23	**Y16**
IO24	**AA17**
IO25	**AA18**
IO26	**AB18**
IO27	**Y18**
IO28	**Y19**
IO29	**AB20**
IO30	**AA20**
IO31	**U16**
IO32	**V16**
IO33	**V17**
IO34	**W16**
IO35	**W17**

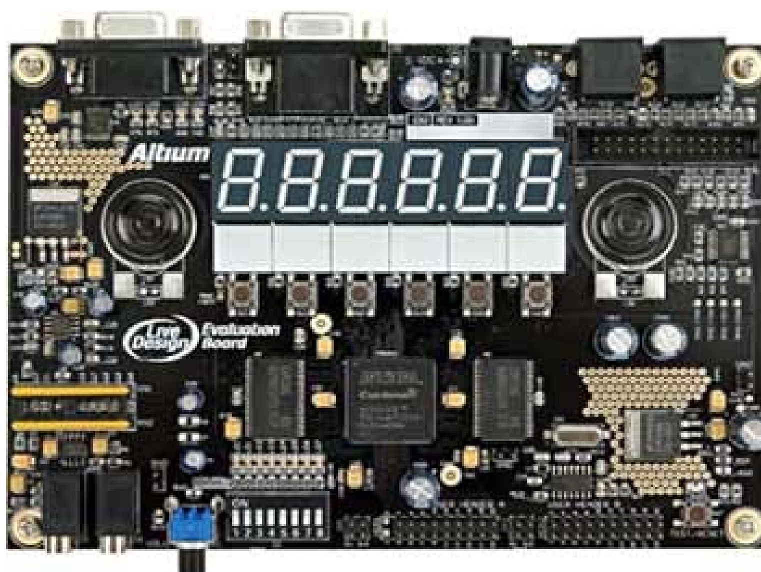
ANEXA 2

Prezenta caracteristicilor sistemului de dezvoltare utilizat în cadrul aplicațiilor din această carte.

Sistemul de dezvoltare cu structura reprogramabilă de tip FPGA Spartan 3 oferă posibilitatea implementării unei game de aplicații destul de variate pentru învățarea şi aprofundarea de către studenți prin descrieri hardware cu ajutorul limbajelor de acelaşi tip a modulelor digitale.

Sistemul de dezvoltare are următoarele caracteristici:

- Este proiectat în jurul structurii reprogramabile de tip SPARTAN 3
- Deține ataşate două memorii statice de tip RAM 256Kx16
- Sistem audio prin implementarea unui modul de tip Delta Sigma, cu două canale şi frecvența de eşantionare ceas ajustabilă având ataşate difuzoare în miniatură cu posibilitatea de control al volumului
- Afişaj pe 6 digiți cu 7 segmente tip LED
- Ceas cu frecvența fixă de 50 MHz
- Port serial RS232, Port VGA, Port PS2 Mini DIN pentru mouse, Port PS2 Mini DIN pentru tastatură, 8 switch-uri DIP, Sistem de 8 leduri
- Doi conectori cu 20 pini I/O şi alimentare pentru extensie
- Buton TEST/RESET definit de utilizator



Sursa de alimentare

Sistemul de dezvoltare este alimentat la 5V în c.c. . Consumul este între 100 şi 500 mA. Dacă puterea totală absorbită de un sistem de dezvoltare cu extensii conectate este mai mare decât cea a sursei standard de 5 volți, atunci se cere o sursă de alimentare de capacitate mai mare.

Acesta, intern are incorporat surse în comutație care furnizează diferite tensiuni de alimentare (+3.3 volți, 1.2 volți şi 2.5 volți).

Circuitele periferice

Pe acest sistem sunt conectate mai multe circuite periferice în vederea utilizării unei game cât mai variate de aplicații. Deține conectori de extensie, memorii, butoane, afişaje cu digiți pe sapte segmente, led-uri, conectori de tip PC/2 pentru conectarea unei tastaturi a unui mouse, conector de tip RS232. Prin cosultarea schemei electrice sau a anexei 1 se pot face uşor legările de interconexiune între circuitele periferice si structura de tip FPGA.

Generatorul de ceas

Sistemul de dezvoltare deține un generator de impulsuri de ceas cu frecvența de 50 MHz. Semnalul de ceas este conectat la circuitul FPGA la pinul **AA12**, numit şi GCLK (ceas global).



Oscilatorul ce cuartz cu frecvența de 50 MHz

Memoria statică RAM

Pe acest sistem sunt plasate două memorii statice de tipul 256k x 16 (SRAM0-U5 şi SRAM1-U6), conectate direct la pinii I/O ai circuitului FPGA. Memoriile SRAM sunt conectate individual la pinii I/O ai FPGA, pentru



ca SRAM-ul să poată fi configurat în mai multe moduri de către FPGA IP (Intellectual Propriety).

Memoria, ca resursă, poate fi configurată de aplicaţie ca un singur spaţiu de 256 x 32, două spaţii de 256 x 16 sau un singur spaţiu de 512k x 8. Timpul de acces este de 10 nS.

Conectorii intrare/ieşire I/O pentru utilizator

Un total de 36 de semnale I/O FPGA sunt conectate la 20 de pini (terminalul A (HDR1) şi terminalul B (HDR2), cu 18 semnale pe fiecare terminal). Aceste terminale au punct de masă şi tensiuni de 3.3 volţi sau 5 volţi, care pot fi selectate prin cei 4 pini disponibili pentru acest lucru JP1 şi JP2.

Terminalul A şi Terminalul B sunt create pentru a permite conectarea la FPGA a altor interfeţe predefinite de utilizator. Pini I/O ai terminalelor pot fi configuraţi ca intrare sau ieşire.

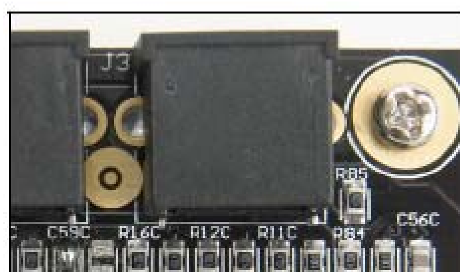
Portul RS232

J4 este portul serial de tipul RS232 care oferă semnale de tipul RXD, TXD, RTS şi CTS. Fiecăruia dintre aceste semnale îi corespunde un led care permite observarea fluxului de date prin acest conector. Nivelele de tranziţie ale portului sunt furnizate de circuitul RS3232.



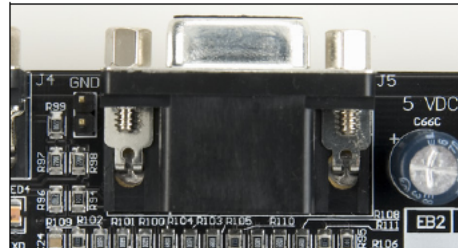
Porturile pentru tastatura şi mouse

J2 şi J3 oferă porturile standard PS2 folosite în mod normal pentru mouse şi tastatură. Pinii acestor porturi sunt legaţii direct la pinii circuitului FPGA.



Portul VGA

J5 (DB15) este un port VGA compatibil monitoarelor video RGB. Portul este configurat cu 3 biți pe culoare cu un total de 9 biți pe pixel sau 512 culori. Portul este conectat direct la 9 pini I/O ai FPGA. Doi pini adiționali I/O asigură semnalele de sincronizare verticală şi orizontală pentru conectorul VGA.



Modulul LED cu 7 segmente

Sistemul de dezvoltare are un display format din 6 digiți a câte 7 segmente. Fiecare segment şi punct decimal este comandat de un pin de ieşite individual al circuitului FPGA.



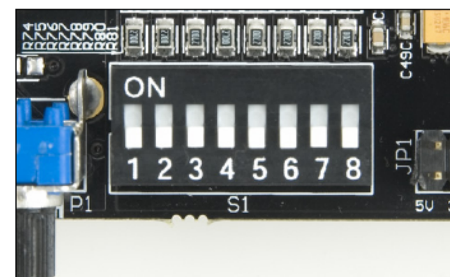
Matricea de taste

Folosind cele 6 taste se poat introduce informații utilizator.. Fiecare buton este conectat la o intrare FPGA.



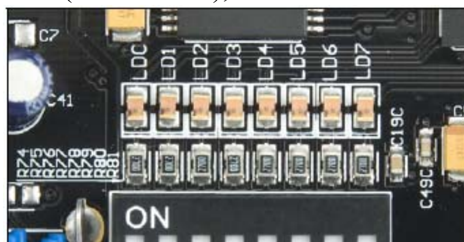
Switch-ul DIP al utilizatorului

Sistemul de dezvoltare are un set de 8 switch-uri conectate fiecare la câte un semnal I/O FPGA. Acestea sunt conectate ca un circuit parțial activ care atunci când fiecare swich este ON, semnalul produs este 0 logic.



Ieşirile de semnal pentru LED-uri

Sistemul de dezvoltare are 8 LED-uri (LD0... LD7), fiecare din ele fiind comandate de un semnal FPGA separat. Semnalele LED-urilor sunt active în 1 logic şi se comandă de către utilizator prin modulul digital proiectat de către acesta.



Pe sistemul de dezvoltare mai are şi un LED notat cu LOADED deasupra butonului de TEST/RESET care se aprinde ori de cate ori structura FPGA a fost configurata corect (LED2).

Butonul definit de utilizator TEST/RESET

Butonul (SW7) etichetat TEST/RESET este conectat la un semnal I/O FPGA. Funcția butonului este determinată în întregime de aplicația utilizatorului, asta înseamnă că nu are nici o funcție intrinsecă până în momentul în care este definită de aplicația utilizatorului.

